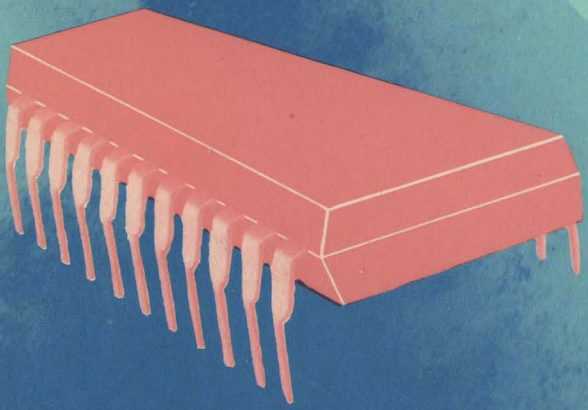
H.-P. Blomeyer-Bartenstein



Mikro Computer Technik

Mikrocomputertechnik

Von Dipl. - Ing., Dipl. Wirtsch. - Ing. H. P. Blomeyer-Bartenstein

ISBN 3-921 682-35-5

Es kann keine Gewähr übernommen werden, daß die in diesem Buche verwendeten Angaben, Schaltungen, Warenbezeichnungen und Warenzeichen frei von Schutzrechten Dritter sind. Alle Angaben werden nur für Amateurzwecke mitgeteilt. Alle Daten und Vergleichsangaben sind als unverbindliche Hinweise zu verstehen. Sie geben in keinem Falle Aufschluß über evtl. Liefermöglichkeiten oder Verfügbarkeiten. In jedem Falle sind die Datenblätter der Hersteller heranzuziehen. Nachdruck und öffentliche Wiedergabe, besonders die Übersetzung in andere Sprachen nur mit vorheriger Genehmigung des Verlages. Irrtum sowie alle Rechte vorbehalten. Beim Betrieb von Sende-und Empfangs-Anlagen sind unbedingt die Bestimmungen der Deutschen Bundespost zu beachten.

Copyright by Ing.W.Hofacker GmbH , 8000 München 75 ,Postfach 75437 . Druck der

1. Auflage im Oktober 1977.

Gedruckt in der Bundesrepublik Deutschland ,printed in West-Germany, Imprimé en RFA

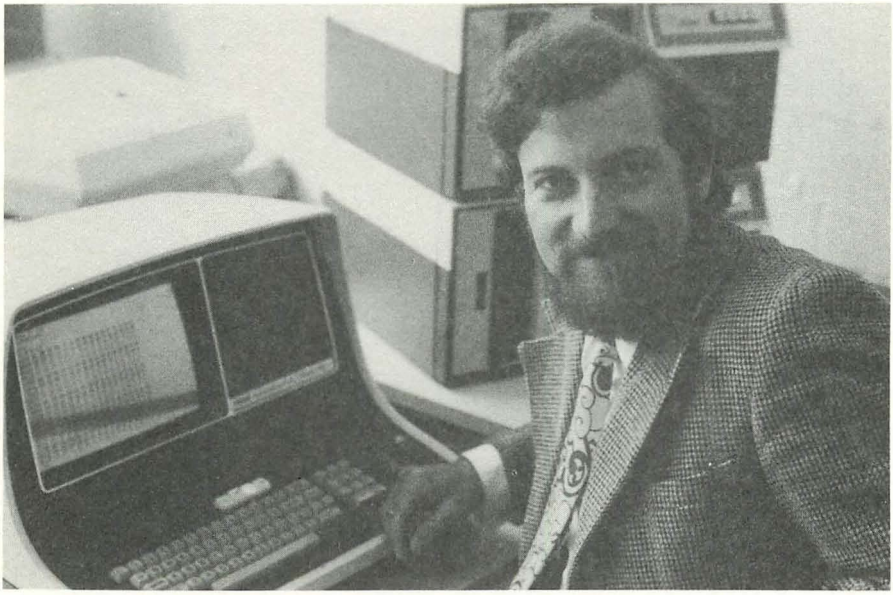
Autorenprofil

Dipl. Ing. Dipl. Wirtsch. Ing. Hans-Peter Blomeyer-Bartenstein (1945) studierte Elektrotechnik und Wirtschaftswissenschaften an der TU München.

Er leitet seit Oktober 1976 den Geschäftsbereich Mikrocomputer der Kontron-GmbH, Eching/München.

Seine Laufbahn begann er im Bauelementebereich der Firma Siemens im Jahre 1970, wo er zunächst DV-Probleme bearbeitete, später das Applikationslabor für Digitalbausteine und zuletzt eine Mikrocomputerapplikationsgruppe leitete; mit Mikroprozessoren befaßt er sich dort seit Ende 1974.

Er hat seitdem eine Reihe Aufsätze in Fachzeitschriften und zwei Bücher über dieses Thema publiziert.



INHALT:

1. Arbeitsweise eines Computersystems 9

1.1.	Was kann ein Computer	9
1.2.	Warum braucht der Geräteentwickler heute Computer-Kenntnisse?	10
1.3.	Struktur eines allgemeinen Computersystems	10
1.4.	Hardware - vom Funtionalelement zum Hilfsmittel	13
1.5.	Software -	13
1.6.	"Computer - Einmaleins": Das binäre und andere Zahlensysteme	15
1.7.	Codes: standardisierte Informationen	19
1.8.	Rechenoperationen im Computer	22
1.9.	Fest– und – Gleitkomma - Arithmetik	25

2. Die Bestandteile von Computersystemen 27

2.1.	Computerstruktur	27
2.2.	Zentralprozessor – "Gehirn" des Computers	29
2.3.	Vom Befehl zum Anwenderprogramm	32
2.4.	Adressierung	35
2.5.	Die Befehlsverarbeitung des Zentralprozessors	40
2.5.3.	Bedingungsbits (="Flag")	45
2.6.	Schlaglichter auf Mikrocomputer–Funktionen	48
2.6.1.	Der "Keller"– oder "Stapel" -Speicher	48
2.6.2.	"Interrupt" und Polling"	52
2.6.3.	Datenübertragung im Quittungsbetrieb: "Handshaking"	56
2.6.4.	Ein/Ausgabe – Schnittstellen	57
2.6.5.	Zähler und Zeitgeber für Echtzeitereignisse	59
2.6.6.	Direkter Speicherzugriff	62
2.6.7.	Periphere Hardware für Mikros	78
2.6.8.	Wahlfreier Zugriff auf Massenspeicher durch Magnetplatten–Technik	80
2.6.9.	In Sache Mikrocomputer -Software	83
2.6.9.1.	Wahl der Programmiersprache	83
2.6.9.2.	Assemblersprache	84
2.6.9.3.	Unterprogrammtechnik	94
2.7.	Die Z 80A–CPU	95
2.8.	Speicher für Programm und Daten	105
2.8.1	Allgemeines	105
2.8.2.	Grundsätzliches zum Anschluß von Festwert– und Schreib/Lese–Speichern	108

2.8.2.1 Minimalkonfiguration 108
2.8.2.2. Zuschaltung statischer RAM's 110
2.8.2.3. Realisierung eines voll ausbaufähigen Computers 110
2.8.2.4. Anschluß "langsamer" Speicherbausteine 113
2.8.2.5. Schaltung zur Speicherbereichs–Auswahl 114
2.8.3. Anschluß von dynamischen Speichern 114
2.8.3.1. CPU -Signale für den Betrieb von dynamischen Speichern 114
2.8.3.2. Grundsätzliche Hinweise zum Anschluß dynamischer Speicherbausteine 117
2.8.3.3. Schaltung mit 22pin–dynamischen 4kbit–RAM–Bausteinen 120
2.8.3.4. Schaltung mit 16pin–dynamischen 4kbit–RAM–Bausteinen 122
2.9. Ein/Ausgabe 125
2.9.1. Interrupt–Architektur 125
2.9.2. Programmierung der Parallel - Ein/Ausgabe (PIO) 131
2.9.3. Programmierung der Zähler- und Zeitgeber-Kanäle 138
2.9.4. Ansteuerung von Datenbus –Puffern 141
2.10. Anwendungsbeispiel für eine Parallel–Schnittstelle: Anschluß eines Thermodruckers 144
2.10.1. Vorbemerkung zum verwendeten Drucker 144
2.10.2. Schaltung der Hardware –Anpassung 144
2.10.3. Zugehöriges Steuerprogramm 146
2.11. Beeinflussung des Programmablaufs über externe Schaltkontakte 150

3. Betriebsbereite Mikrocomputersysteme 151

3.1. OEM–Mikrocomputersystem mit eingebautem Massenspeicher 152
3.1.1. Systemkonzept 152
3.1.2. Beschreibung der Anlage 152
3.1.3. Erweiterungsmöglichkeiten 153
3.1.4. Das System Z80–PDS 154
3.2. Mikrocomputersysteme zur Lösung einfacher und kosten–kritischer Probleme 154
3.2.1. Systemeigenschaften 155
3.2.2. Erweiterungsmöglichkeiten 155
3.2.3. Bedienungs– und Anzeige–Einheit 157
3.2.4. Bedienungskonzept 158
3.2.5. Kommandosprache 160
3.2.5.1. Setzen von Registern auf einen gewünschten Wert 160
3.2.5.2. Anzeige von Registerinhalten 161
3.2.5.3 Eingeben eines Wertes in eine Speicherzelle 161
3.2.5.4. Ausgabe des Inhaltes einer Speicherzelle 162
3.2.5.5. Eingabe eines Anwenderprogramms 162
3.2.5.6. Ausgabe von Speicherbereichen 164
3.2.5.7. Starten eines Anwenderprogramms 165
3.2.5.8. Programmausführung im Einzelschrittverfahren 165
3.2.5.9. Setzen und Löschen eines Haltepunktes 165

4. Entwicklung eines Mikrocomputer -Anwender -Systems 166

4.1. Entwicklungsziele 166
4.2. Wie geht man bei der Entwicklung vor? 167
4.3. Praktische Beispiele einer Programmentwicklung 180
4.3.1. Spielprogramm 180
4.3.2. Multiplikation 189
4.3.3. Beispiel eines Makroprogramms 191
4.3.4 Prüfprogramm für den Z80–KIT 195
4.3.5. Beispiel für einen Programmentwicklungsvorgang 198
4.4. Programmierung des Mikrocomputer -Programmspeichers 204

5. Mikrocomputer -Entwicklungssystem 205

5.1. Mindestausstattung 205
5.2. Erweiterungen 208

6. Technische Einzelheiten der Entwicklungssystem –Komponenten 210

6.1. Zentralgerät 211
6.2. Zusätzliche Steckkarten 213
6.3. Floppy–Disk–Einheit 214
6.4. Systemkonsole 214
6.5. Echtzeittestadapter 214
6.6. Schnelldrucker 214
6.7. PROM–Programmiergerät 215

7. Software für Mikrocomputer–Entwicklungssysteme und OEM-Computer 215

7.1. Editor 216
7.2. Datenverwaltung ("File Maintenance") 217
7.3. Assembler 218
7.4. Debug 218
7.5. Übersetzungsprogramme für höhere Programmiersprachen 220

8. Beurteilung und Bewertung von Mikrocomputern 221

1. Arbeitsweise eines Computersystems

1.1. Was kann ein Computer?

Nach mehr als einem viertel Jahrhundert Computertechnik hat sich - gottseidank - eine gewisse Skepsis durchgesetzt. Man weiß nun, daß die vielbenützten Stereotypen "Es geht jetzt alles etwas langsamer, wir haben auf Datenverarbeitung umgestellt" oder "der Computer hat Ihren Kontostand leider falsch ausgespuckt" nicht auf Computer-Psychatrie, sondern schlicht auf Schlamperei der Programmierer und des Bedienpersonals zurückzuführen sind, denn elektronische Rechenanlagen arbeiten

- wesentlich schneller,
- genauer und
- mit einer höheren Fehler–Unwahrscheinlichkeit

als andere Datenverarbeitungsverfahren und sogar als das menschliche Gehirn.

Genau diese Fähigkeiten machen die Leistungsfähigkeit und die Wirtschaftlichkeit dieser Maschinen aus.
Anders als bei den Kybernetik–Monstern aus Science–Fiction–Filmen sind ihre Möglichkeiten jedoch klar eingeschränkt:

- ☐ Computer können **nur die** Operationen ausführen, für die sie vorher "programmiert" wurden.
- ☐ Jede einzelne Aktivität muß also vorher vom Menschen – dem Entwicklungsingenieur – bereits logisch vorvollzogen sein; die Maschine kann sie dann jedoch wesentlich rascher und beliebig oft durchführen.

In dieser Aussage ist impliziert, daß Computer nicht "denken" oder "fühlen" können – wozu Kreativität, freie Assoziation und Lernfähigkeit gehören, Dinge, die durch die oben beschriebene deterministische Verhaltensweisen von Maschinen ausgeschlossen sind.

Zwar bemüht sich die kybernetische Wissenschaft, diesen Vorgängen auf die Spur zu kommen und sie technisch zu reproduzieren, aber bis zu einer technologischen Implementierung ist es sicher noch ein sehr weiter Weg.

Eine neue Wundergläubigkeit hat nun das Erscheinen des Mikrocomputers ausgelöst: "Das Rechenzentrum in der Streichhholzschachtel" – oder so ähnlich.

Tatsächlich arbeiten Mikrocomputer auf dem gleichen Prinzip wie die "klassischen" Groß– oder Mini–Rechner, und nur der ungeheuer rasche Fortschritt auf dem Gebiet der Halbleitertechnologie hat die Integration von Rechnerfunktionen auf einem einzigen Silizium–Chip in wirtschaftlicher Weise ermöglicht.

Man muß sich jedoch darüber klar sein, daß ihre Fähigkeiten genauso beschränkt sind wie die herkömmlicher Computer, im Gegenteil: Der erfahrene Computeranwender vermißt bei den "Mikros" einen gewissen "Komfort".

1.2. Warum braucht der Geräte-Entwickler heute Computerkenntnisse?

Der Mikrocomputer–Technik gehört die nächste Zukunft – diese Aussage ist inzwischen unbestritten und wird täglich durch die Entwicklung neuartiger Mikrocomputer-Anwendungen und das rapid wachsende Volumen an verkauften Mikrocomputer–Bausteinen bestätigt.

Grund hierfür sind bestimmte Eigenschaften von Mikrocomputersystemen, die deren Einsatz für einen sich dauernd erweiternden Anwenderkreis interessant machen, nämlich

- Die enorme Verkürzung der Gesamt–Entwicklungsdauer bei Mikrocomputer–Anwendersystemen gegenüber herkömmlichen Digitalschaltungen.
  Dadurch ist es möglich, neue Produkte rascher auf den Markt zu bringen.
- Änderungs– und Erweiterungs–Möglichkeit, die bei Mikrocomputersystemen wesentlich höher ist als bei anderen Systemen, da bei Mikrocomputersystemen in vielen Fällen der Austausch einiger weniger Programmspeicherbausteine genügt, um die gesamte Gerätefunktion zu ändern.
- die Wartungsfreundlichkeit durch Eigen–Tests, die im Anwenderprogramm eingebaut werden können und
- die vielfältigen neuen technischen Möglichkeiten, die der Einbau vom Computer–"Intelligenz" in Geräte mit sich bringt, die uns täglich umgeben.

In dem eben Gesagten ist schon beinhaltet, daß die Mikrocomputer–Technik im Allgemeinen nicht (wie das bei den älteren Geschwistern der Mikrocomputer, nämlich den Mini– oder Großcomputern der Fall ist) darin besteht, mit Hilfe verschiedenartiger Programme auf der gleichen Anlage (="Hardware") mathematische oder kommerzielle Daten zu verarbeiten und Probleme zu lösen.
Stattdessen **dient** der Mikrocomputer im allgemeinen dazu, in einem bestimmten Gerät die **eine** Aufgabe zu erfüllen, die dieses Gerät hat, also die entsprechende **Gerätefunktion zu realisieren,** und zwar **anstelle** einer Realisierung mit herkömmlicher, festverdrahteter Elektronik, Pneumatik oder Mechanik.

Will nun der Geräte-Entwickler, der seine Problemlösungen bisher eben in konventioneller Weise ausarbeitete, die vorher genannten Vorteile der Mikrocomputer–Technik ausnutzen, muß er in der Computerentwicklung zuhause sein, da Mikrocomputer nach dem gleichen Prinzip arbeiten wie andere Computer.

Die Notwendigkeit, Computerentwicklung zu betreiben, kann also nicht wie bisher auf den relativ kleinen Kreis der Ingenieure beschränkt bleiben, die in der Datenverarbeitung tätig sind.

1.3. Struktur eines allgemeinen Computersystems

Das Wort "System" ist eines der abgegriffensten im technischen Sprachverkehr. Wir wollen es trotzdem benützen, weil es sehr praktisch ist, wenn man es nur klar definiert:

Unter einem System versteht man ein strukturiertes Gebilde, das Ausgaben (="Outputs") aufgrund von Eingaben (="Inputs") zeitigt. Welche Outputs aufgrund welcher Inputs erzeugt werden, ist durch seine interne Struktur bestimmt (Bild 1).

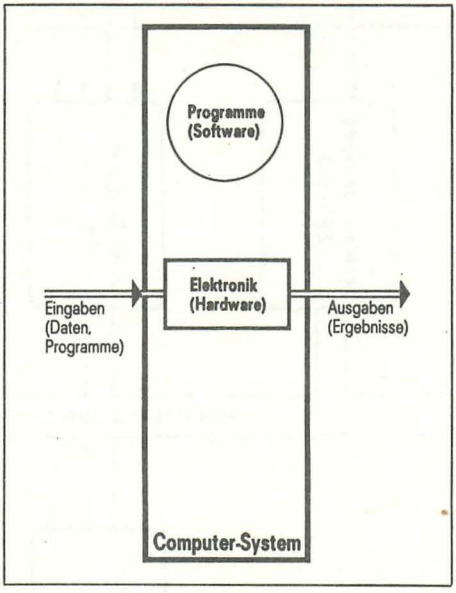


Bild 1 Hardware und Software in einem Computersystem

Das Gesamtsystem kann man in eindeutiger Form beschreiben und in Sub–Systeme untergliedern (="Systemanalyse"), bis man auf der Realisierungsebene (z.B. in elektronischen Systemen dem integrierten Schaltkreis) angelangt ist.

Diese Betrachtung gilt für alle Arten von Systemen. Wenn wir es mit Computersystemen zu tun haben, besteht das Ganze im Wesentlichen aus zwei Hauptkomponenten: Der Elektronik zuzüglich der zugehörigen Elektromechanik (="Hardware") und dem Programm (="Software").

Wenn man etwas als ein "Computersystem" bezeichnet, will man also ausdrükken, daß man es mit einer voll funktionsfähigen Einheit zu tun hat, die Hardware (inklusive der Möglichkeiten zur Ein/Ausgabe von Informationen, der sogenannten Peripherie) **und** Software umfaßt (Bild 2).

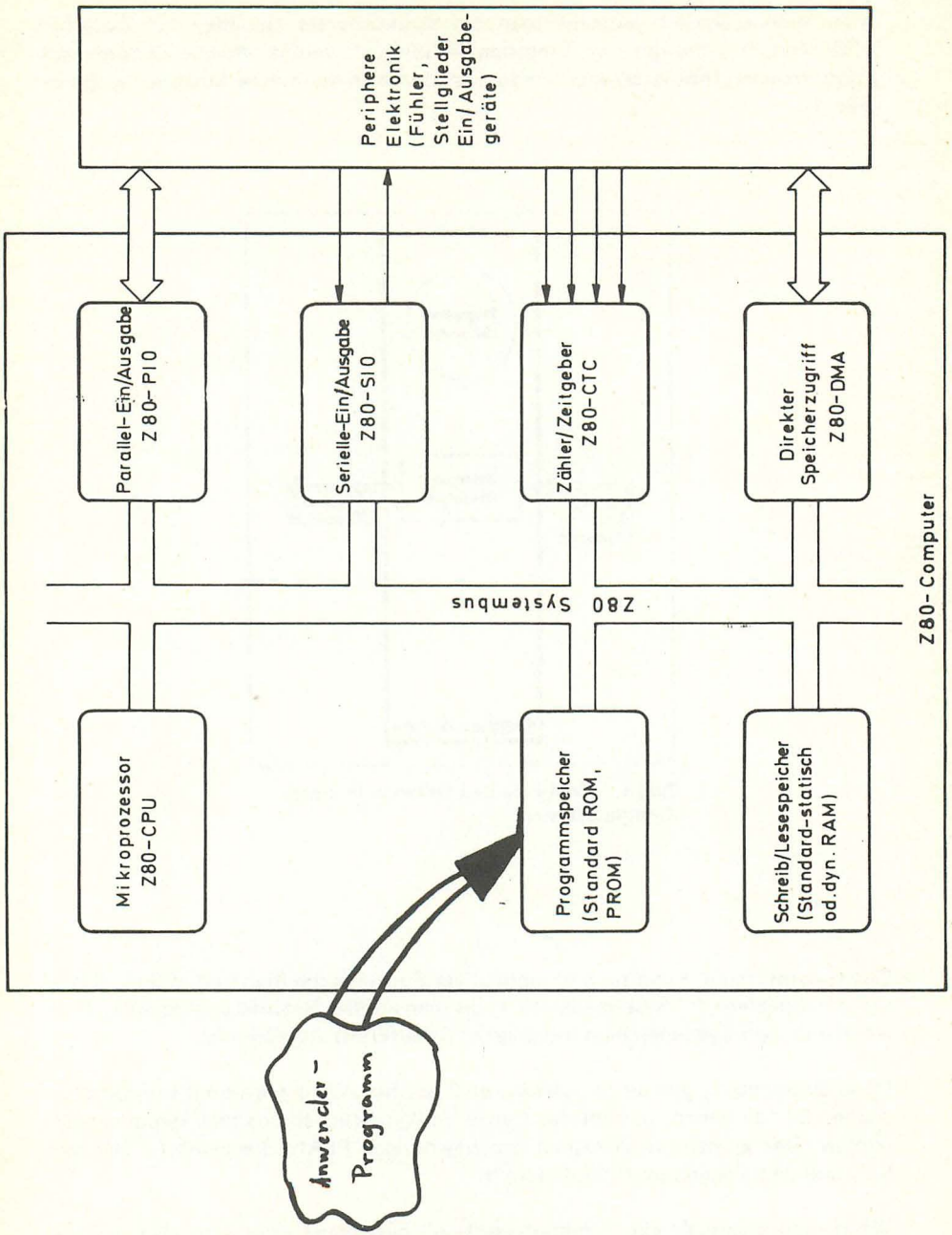


Bild 2: Beispiel eines Mikrocomputersystems

1.4. Hardware - vom Funktionalelement zum Hilfsmittel

Ein elektro-mechanisches System (z.B. ein mechanischer Tischrechner oder die konventionelle Zündung in einem Auto) bestand ausschließlich aus Hardware*.
Das gleiche gilt auch für "klassische" (also nichtprogrammierbare) elektronische Systeme.
Ihre Eingaben beschreiben irgendwelche physikalischen Vorgänge mit Hilfe der Elektrotechnik, und zwar in analoger oder digitaler Form.
Im letzten Fall sind als Informationseinheiten nur noch die Signalpegel LOW und HIGH (entsprechen dem logischen JA und NEIN, WAHR und FALSCH, EIN und AUS), die sich halbleitertechnisch und logisch besonders leicht beherrschen lassen.
Immer war hier die Funktion (also was das Gerät aufgrund welcher Eingangsbedingungen tun muß) durch die interne Verknüpfung der einzelnen Funktionselemente gegeben.
Da diese Verknüpfung bei der Elektronik mit Drähten erfolgt, spricht man bei nichtprogrammierbaren Digitalschaltungen daher von "Festverdrahteter Logik" oder "Hardwired Logic".

Der Wunsch, Änderungen leichter durchführbar zu machen und die Tatsache, daß

- die Funktion eines Digitalgatters, woraus festverdrahtete Logik im Prinzip aufgebaut ist, sich immer durch einige wenige Speicherstellen in einem programmierbaren Gerät ersetzen läßt und
- die Kosten eines Digitalgatters viel höher sind als die Kosten der äquivalenten Anzahl der Speicherzellen.

hat rasch dazu geführt, statt einem festverdrahteten Gerät eine Universallogik zu realisieren, deren Funktion endgültig erst durch Vorgabe eines Programms festgelegt wird (="Computer").

Die Hardware ist nur noch Hilfsmittel, die aus wenigen hochintegrierten Standardbausteinen realisiert wird und auf der das Programm "abläuft".

1.5. Software

Das (oder die) Programme besteht aus einer Folge binärer Ziffern, also Nullen und Einsen, die die Funktionen des Computers und damit des gesamten Gerätes kontrollieren; wie das in Einzelnen vor sich geht, wollen wir in Abschnitt 2.1. näher betrachten.

* Das technische Wissen, das in einem System steckt, ist übrigens nie Software, denn wir haben vereinbart, Software als Funktionalelement aufzufassen. All die Dinge, die im fertigen System nicht mehr in Erscheinung treten, aber zur Realisierung nötig waren (z.B. Schaltpläne, Know-How) wollen wir als BRAINWARE bezeichnen.

Zunächst soll aber die Systematik hier weiterverfolgt und die Begriffswelt geordnet werden. Das Programm, das die Gerätefunktion erfüllt, heißt in der Mikrocomputer–Technik ANWENDERPROGRAMM. In Zeitschriften und Prospekten findet man stattdessen manchmal das Wort "Mikroprogramm" – was erstens unsinnig ist, da an den immer komplexer werdenden Mikrocomputer–Anwenderprogrammen nichts mehr "Mikro" ist, und zweitens verwirrend, da man darunter in der Datentechnik ein Programm aus Mikroinstruktionen versteht, die direkt auf die logischen Funktionen des Zentralprozessors wirken – also das Programm, das die Maschinenbefehle auf Gatterebene interpretiert. Nebenbei gesagt, gibt es tatsächlich Maschinen (auch Mikroprozessoren), bei denen man Zugriff auf das Mikroprogramm hat; diese Maschinen nennt man dann MIKROPROGRAMMIER–BAR. Bei den weitverbreiteten Mikro–Computern in nMOS–Technologie, mit denen wir uns hauptsächlich befassen, ist das jedoch nicht der Fall.

Von den Anwenderprogrammen zu unterscheiden sind die **Systemprogramme**. Dabei handelt es sich um Programme zur Bearbeitung von Verwaltungsaufgaben auf einem Universalrechner. In unserem spezifischen Anwendersystem, in dem ein einziges fest vorgegebenes Programm arbeitet, gibt es nichts zu verwalten und folglich auch keine Systemprogramme.

Da andererseits **Mikrocomputer -Entwicklungssysteme** eine Art Universalrechner sind, gibt es auch in der Mikrocomputer–Technik Systemprogramme. Sie befähigen den Mikroprozessor, als Mikrocomputer–Entwicklungshilfe eingesetzt zu werden.
Mikrocomputer–Systemprogramme sind zum Beispiel

- ASSEMBLER - Programm
- PL/Z–Compiler
- Fehlersuch–Programm
- FD–Dateiverwaltungs -Programm
- Texteditor

Näheres hierüber werden wir bei der Besprechung des Entwicklungssystems erfahren.

In Absatz 1.2. wurde schon angedeutet, daß das Anwenderprogramm in der Lebensdauer des Gerätes meist gleich bleibt; es lohnt sich also nicht oder ist sogar unerwünscht, dieses Programm bei Bedarf immer wieder in den Schreib/Lesespeicher des Mikrocomputers einzulesen, wie das bei konventionellen Rechenanlagen der Fall ist. Stattdessen stellt man es dem Zentralprozessor in Festwertspeicher–Bausteinen zur Verfügung, wo das Programm auch bei Stromversorgungsausfall nicht verloren geht (Abschnitt 2.5.3.).

Solche in Hardware (z.B. in Festwertspeicherbausteinen) untergebrachte Software bezeichnet man als FIRMWARE.

1.6. "Computereinmaleins": Das binäre und andere Zahlensysteme

Wir wollen nun die untersten Grundelemente der Software etwas genauer ansehen: Die Binärziffern und die aus ihnen gebildeten Binärzahlen.
Es wurde bereits erwähnt, daß die binäre Zahlendarstellung in Computern fast ausschließlich benützt wird, da die Unterscheidung "Signal vorhanden" (=binäre Eins) und "Signal nicht vorhanden" (=binäre Null) sich elektronisch besonders leicht handhaben läßt.

Das Zahlensystem*, das auf Binärziffern basiert, heißt "Dualsystem" oder "Binärsystem"; mathematisch gesprochen, hat es die Zahl Zwei zur Basis, d.h. Zahlen werden im Dualsystem als eine Summe von ganzzahligen Potenzen der Zahl Zwei dargestellt. Im Dezimalsystem werden Zahlen analog als Summe ganzzahliger Potenzen der Zahl Zehn dargestellt; machen Sie sich bitte klar, daß die Dezimalzahl Zweihundert und Neunzehn (geschrieben 219Dez) nichts anderes bedeutet als

$$219_{DEZ} = (2 \times 10^2 + 1 \times 10^1 + 9 \times 10^0)_{DEZ}$$

Ziffer — Wertigkeit; Ziffer — Wertigkeit; Ziffer — Wertigkeit

Will man in einer Maschine mit dieser Zahl etwas anfangen, so muß sie auf jeder Leitung immer mindestens die Ziffern Null bis Neun (also 10 Ziffern) sicher unterscheiden können, andernfalls kann sie kein eindeutiges Ergebnis bringen, weil sonst Grundeinheiten des Zahlensystems (eben Ziffern) nicht voneinander unterschieden werden könnten.

Hat man dagegen lediglich zwei Ziffern zur Zahlendarstellung zur Verfügung, wie das bei elektronischen Rechenanlagen ebenso wie bei Mikro–Computern sinnvoll ist, muß man die ganze Zahl in Vielfache von Potenzen von Zwei aufspalten; unser voriges Beispiel ergäbe

$$219_{DEZ} = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Binär-Ziffern

Machen Sie sich die Arbeit und führen Sie zu Kontrollzwecken die Rechnung selbst durch:

* Weiterer Beitrag zur Sprachverwirrung: Beachten Sie bitte, daß hier das Wort "System" nicht im exakten systemtheoretischen Sinne gebraucht wird, sondern für "Systematik".

Sie erhalten:

$$
\begin{array}{rl}
2^7 = & 128 \times 1 = 128 \\
2^6 = & 64 \times 1 = 64 \\
2^5 = & 32 \times 0 = 0 \\
2^4 = & 16 \times 1 = 16 \\
2^3 = & 8 \times 1 = 8 \\
2^2 = & 4 \times 0 = 0 \\
2^1 = & 2 \times 1 = 2 \\
2^0 = & 1 \times 1 = 1 \\
\hline
 & + \ 219
\end{array}
$$

Für Mathematik-Fans unter Ihnen:
Die allgemeine Formel für die Darstellung positiver, ganzer Zahlen in einem beliebigen Zahlensystem zur Basis b lautet: $x = \sum_{v} (k_v \times b^v)$

mit $v = o \ldots \infty$, wobei die Konstanten ganzzahlig und $0 \leq k_v < b$ sein müssen.
Wir werden diese Verallgemeinerung später noch für andere Zahlensysteme brauchen.
Hier zunächst noch für den Praktiker die Rechenregeln für die Umwandlung zwischen Dezimal- Und Dualzahlen.

Zur Verwandlung einer Dezimalzahl in eine Dualzahl bieten sich zwei Methoden an:

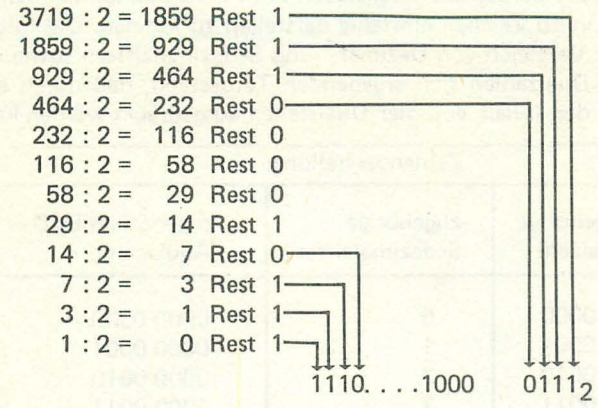
1) die Dezimalzahl wird in eine Summe von Zweierpotenzen zerlegt. Dazu wird die Dezimalzahl als Summe der größten in ihr enthaltenen Zweierpotenz und als Rest dargestellt. Mit dem jeweils verbleibenden Rest wird entsprechend verfahren. Zuletzt werden diejenigen Stellen, bei denen aktuelle Stellenwerte in dieser Summenzerlegung nach Zweierpotenzen auftreten, mit den Stelleninhalten 1, die übrigen mit 0 besetzt.

Beispiel: $3719_{10} = 2048 + 1671$
$= 2048 + 1024 + 647$
$= 2048 + 1024 + 512 + 135$
$= 2048 + 1024 + 512 + 128 + 7$
$= 2048 + 1024 + 512 + 128 + 4 + 3$
$= 2048 + 1024 + 512 + 128 + 4 + 2 + 1$
$\triangleq 1110 \ \ 1000 \ \ 0111_2$

2) Allgemein (d.h. für beliebige Zahlensysteme anwendbar) und leichter handhabbar ist folgende Konvertierungsregel:

Man dividiert die Dezimalzahl durch die Basis des gewünschten Zahlensystems. Das **ganzzahlige** Divisionsergebnis wird erneut dividiert bis das Endergebnis Null erbringt. Auftretende Divisionsreste bilden den Aufbau der gesuchten Zahl, beginnend von **rechts.**

Beispiel:



Zur Umwandlung aus einem beliebigen Zahlensystem in das Dezimalsystem benützt man vorhergenannte Formel oder aber im Spezialfall der Umwandlung von Dual– in Dezimalzahl das folgende Verfahren nach HORNER:

- Als Kopfzeile wird die Dualzahl aufgetragen. Als erste Spalte der zweiten Zeile steht immer eine Null. Links vor dem Schema erscheint vor der Null eine 2.
- Die Ziffern der ersten Spalte werden addiert.
- Das Ergebnis wird mit 2 multipliziert und in die zweite Zeile der zweiten Spalte geschrieben.
- Die Zeilen der zweiten Spalte werden addiert und das Verfahren bis zur letzten Spalte fortgesetzt.
- Ganz **rechts** erscheint die der Dualzahl äquivalente Dezimalzahl.

Beispiel:

	111	0	1	0	0	0	0	1	1	1
	2026	14	28	58	116	232	464	928	1858	3718
	137	14	19	58	116	232	464	929	1859	3719

Außer dem dezimalen Zahlensystem, das unserer Denkweise angepaßt ist, und dem binären Zahlensystem, das der Zahlendarstellung in Computern gerecht wird, ist noch ein Kompromiß gebräuchlich, der einerseits das Hinschreiben 'vieler' Nullen und Einsen umgeht, andererseits zum Dualsystem kompatibel ist, nämlich alle Potenzen der Zahl Zwei voll ausnützt (das Dualsystem "paßt" nicht so recht mit dem Binärsystem zusammen, da $2^3 = 8$ seinen vollen Ziffernvorrat, nämlich 10, noch nicht ausschöpft, während die nächsthöhere Potenz $2^4 = 16$ seinen Ziffernvorrat von 10 bereits überschreitet).

Dieses Zahlensystem, das bei Mikrocomputern hauptsächlich zur Darstellung von Zahlen auf Maschinenebene benützt wird, heißt Sedezimal– oder Hexadezimal–System (Labor–Slang: "Hex").

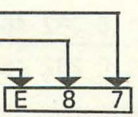
Die ersten 10 Ziffern des Sedezimalsystems lauten wie im Dezimalsystem 0... 9; für die restlichen 6 Ziffern werden die Buchstaben A ... F verwendet, um den gesamten Ziffernvorrat von 16 Zeichen **einstellig** darstellen zu können. Die folgende Tabelle 1 zeigt den Vergleich von Dezimal– und Sedezimalziffern sowie den nach Konvertierung in Dualzahlen sich ergebenden Tatbestand, daß durch eine einzige Sedezimalziffer der Inhalt von vier Dualstellen ausgedrückt werden kann.

Tabelle 1: Zahlendarstellung

Dezimalzahl	zugehörige Dualzahl	zugehörige Sedezimalziffer	zugehöriges BCD–Äquivalent
0	0000	0	0000 0000
1	0001	1	0000 0001
2	0010	2	0000 0010
3	0011	3	0000 0011
4	0100	4	0000 0100
5	0101	5	0000 0101
6	0110	6	0000 0110
7	0111	7	0000 0111
8	1000	8	0000 1000
9	1001	9	0000 1001
10	1010	A	0001 0000
11	1011	B	0001 0001
12	1100	C	0001 0010
13	1101	D	0001 0011
14	1110	E	0001 0100
15	1111	F	0001 0101

Entsprechend der angegebenen allgemeinen Konvertierungsregel läßt sich die Umrechnung der Dezimalzahl 3719_{10} in eine Sedezimalzahl vornehmen:

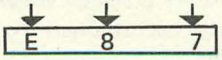
Beispiel:
3719 : 16 = 232 Rest 7
232 : 16 = 14 Rest 8
14 : 16 = 0 Rest 14



(16)

Das gleiche Ergebnis wird erzielt, wenn die der Dezimalzahl 3719_{10} entsprechende Dualzahl in Vierergruppen eingeteilt und das dafür jeweilige sedezimal Äquivalent angeschrieben wird:

Beispiel: $3719_{10} = 1110\ 1000\ 0111_2$



(16)

Die Zusammenfassung von vier Dualstellen heißt Tetrade. Die Darstellung einer Tetrade durch eine Sedezimalziffer ist deshalb möglich, weil sich in vier Dualstellen genau 16 verschiedene Binärkombinationen und damit die Ziffern (0 ... 9, A... F) erzeugen lassen.

Die Dekonvertierung erfolgt entweder nach dem Hornerschema:

	E	8	7
16	0	224	3712
	14	232	3719_{10}

oder nach der Stellenwertregel:

$$E87_{16} = 14 \times 16^2 + 8 \times 16^1 + 7 \times 16^0 =$$
$$= 14 \times 256 + 8 \times 16 + 7 \times 1 = 3719_{10}$$

Aus der Rechnertradition ist übrigens auch das Oktalsystem (Basis = 8) verbreitet. Da moderne Mikrocomputer jedoch hauptsächlich 8 oder 16–bit–orientiert sind, ist eine Aufteilung der Zahlen in 3bit–"Pakete", wie das beim Oktalsystem der Fall ist, sinnlos und daher nicht üblich.

1.7. "Codes" oder standardisierte Informationen

Wir haben bereits klargestellt, daß der Mikrocomputer Informationen grundsätzlich in binärer Form - d.h., soweit es sich um Zahlen handelt, in Dualsystem vermittelt bekommt, man spricht dabei auch von "Binär-Code".

Zur Vereinfachung der Mensch–Maschine–Kommunikation bedient man sich jedoch häufig auch anderer Codes, die mehrere Binärziffern zu einem Codezeichen zusammenfassen; man muß dann z.B. bei einer Eingabe per Tastatur statt mehreren "Nuller"–Tasten und "Einser"–Tasten nur noch eine Taste drücken, die mehrere Binärziffern repräsentiert. Da der Rechner grundsätzlich binär arbeitet, ist eine Umwandlung vom verwendeten Code in den Binärcode nötig. Diese Umwandlung geschieht manchmal über elektronische Schaltungen (also per Hardware), meist jedoch über ein Programm (also per Software).

Ein Beispiel für einen solchen Code haben wir bereits kennengelernt: Werden einem Computer Hexadezimalzahlen eingegeben, so geschieht das eben im zugehörigen Code, dem "Hex–Code", der aus Gründen der einfachen Darstellung identisch ist mit dem binären Hexadezimalzahlen–Äquivalent.
Welche Binärzahlen den 16 Zeichen des Hexadezimal–Codes entsprechen, haben wir in Tabelle 1 unter Abschnitt 1.6. bereits gesehen.

In vielen Mikrocomputer–Anwendungen sind Binärcode oder Hexadezimalcode (der ja nichts anderes ist als eine abgekürzte Schreibweise des Binärcodes, in dem der Zentralprozessor direkt arbeitet) zu unhandlich, nämlich meist dann, wenn über die Mensch/Maschine - Schnittstelle Zahlen in den für Menschen lesbarer Form transportiert werden.
Man stelle sich das (stark vereinfachte) Beispiel eines Mikrocomputer-gesteuerten Schrittmotors vor (Bild 3), dessen Ziel Position in Millimetern an Codier– oder Vorwahlschaltern über mehrere Stellen dezimal eingestellt und dessen Ist–Posi-

tion über Sieben–Segment ebenfalls dezimal ausgegeben wird. Die Anzahl der auszuführenden Schritte ist aus der Differenz zwischen Soll– und Ist–Position zu ermitteln.

Hier möchte man natürlich die von einer Bedienperson einzugebende SOLL–POSITION als Dezimalzahl vorgeben und auch die Anzeige der momentanen IST – POSITION als Dezimalwert ablesen können.
In dieser Darstellung ist die Dezimalziffer die Einheit; uns eine digitaltechnische Behandlung dieses Problems möglich zu machen, muß man demzufolge jede dieser Dezimalziffern in binärer Form darstellen. (="codieren"). Dies geschieht beispielsweise durch sogenannte "Codierschalter", wie dies im Bild 3 dargestellt ist, auf elektromechanischem Wege oder durch entsprechend ausgelegte Digital–Dekodier–Bausteine oder aber durch ein geeignetes Mikrocomputerprogramm.

So sind beispielsweise in der Z80–CPU* eine Reihe solcher Sonder–Instruktionen zur einfachen und raschen Verarbeitung von 4bit–Wörtern implementiert; wir werden sie bei der Behandlung des Mikrocomputer–Befehlsatzes noch näher besprechen.

Diese Art der Dezimalzahlendarstellung durch Binäräquivalente nennt man BCD–Code (="**B**inär **C**odierte **D**ezimal–Ziffer").

Zur Codierung dienen einfach die ersten 10 Wörter (0 ... 9) des Dezimalcodes; diese Codierung ist in Tabelle 1 gezeigt. Hier wird darüberhinaus deutlich gemacht, welcher Unterschied zwischen dem Binäräquivalent einer Dezimalzahl (z.B. 14) und dessen BCD–Code besteht; dies kommt ganz einfach daher, daß der BCD–Code nur "bis 9 geht", und man einer Ziffer genau **ein** binäres 4bit–Codewort zuordnet, in 4 Binärstellen jedoch die Dezimalzahlen 1 ...15 "hineinpassen", wodurch die Gleichheit von Binär– und BCD–Code jenseits der Zahl neun verloren geht.

* Z80–CPU ist der Mikroprozessor der Z80-Mikrocomputerserie der Fa. ZILOG Inc.

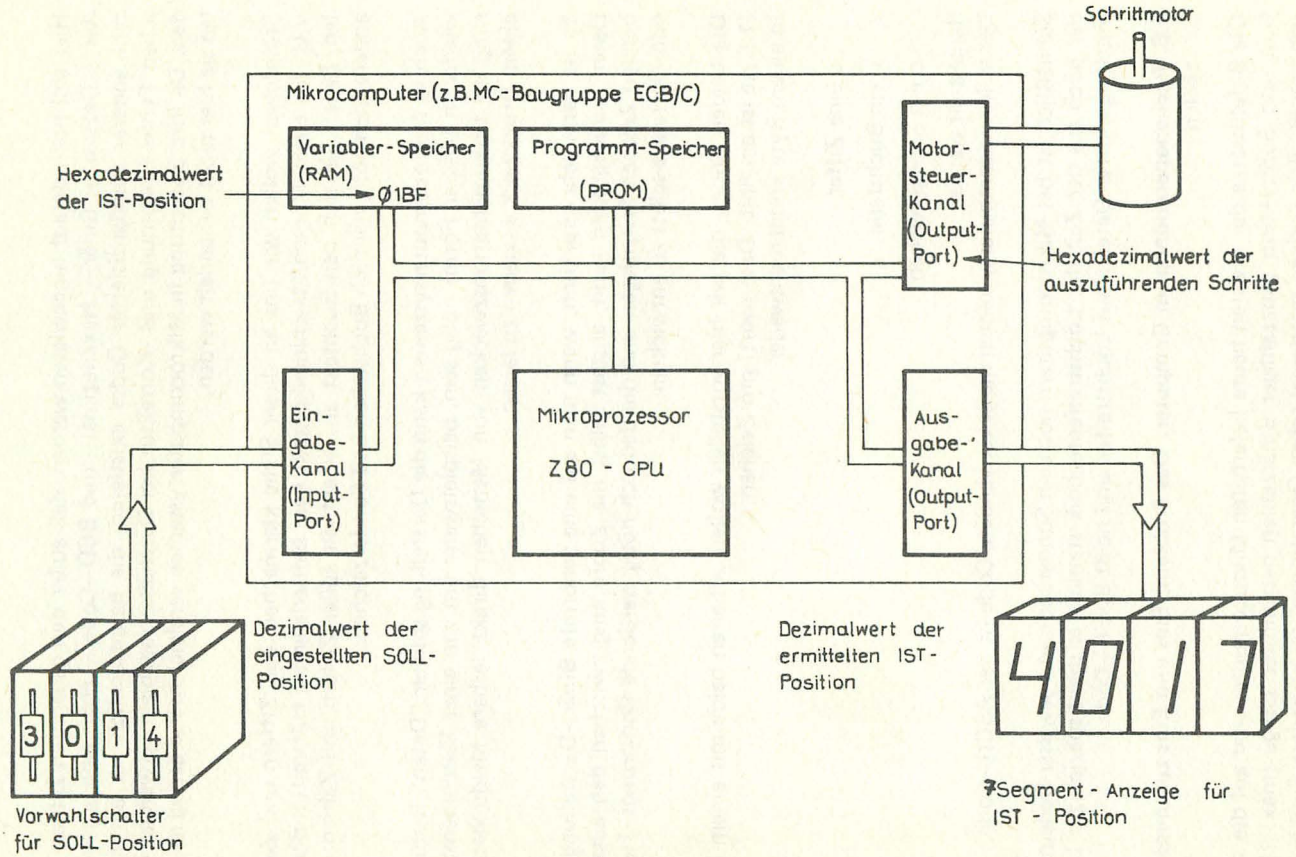


Bild 3: Praktisches Beispiel eines Mikrocomputersystems

Hier soll noch darauf hingewiesen werden, daß außer den genannten Zahlen–Codes – Dezimal–, Binär–, Hexadezimal– und BCD–Code – noch eine große Anzahl anderer standardisierter Codes existieren, die hauptsächlich zur automatischen Fehler-Erkennung und -Korrektur bei Datenübertragungen verwendet werden. Da ihre Bedeutung in Mikrocomputersystemen vergleichsweise gering ist, sollen sie hier nicht behandelt werden.

Stattdessen wollen wir uns an dieser Stelle klarmachen, daß Ziffern und Zahlen u.U. für einen Mensch/Maschinenkontakt kein befriedigendes Hilfsmittel darstellen. Sehen Sie sich den Buchtext an: er enthält außer Ziffern und Zahlen noch andere Elemente, nämlich Buchstaben und Sonderzeichen.

Auch in Mikrocomputersystemen kann die Darstellung solcher "Daten" (natürlich wieder in binärer Form) nötig sein, beispielsweise um von einer Maschinensteuerung zu Dokumentationszwecken von Menschen direkt lesbare Meldungen auf einem Druckwerk ausgeben zu lassen.

Es ist dann sehr praktisch, wenn man auf eine genormte Binär–Darstellung von Daten zurückgreifen kann, anstatt selbst die Zuordnung zwischen den auszugebenden Zeichen festzulegen und die dadurch nötig werdende Codierungs–Hard– oder Software selbst zu entwickeln.

Die beiden Codes, die wir hier vorstellen wollen, arbeiten beide mit einem Byte (1 Byte ist ein 8bit–Datenwort) pro Zeichen.
In einem Byte können entweder

- o eine Ziffer
- o ein Buchstabe
- o ein Sonderzeichen

dargestellt werden.
Der in Mikrocomputersystemen meist verwendete Code ist der ASCII–Code.

Schließlich ist bei Mikrocomputern noch ein Sondercode von gewisser Bedeutung, der noch aus der Zeit der Lochstreifentechnik stammt; er dient speziell zur Übermittlung binärer Dateien auf Lochstreifen und heißt BNPF–Code.

1.8. Rechenoperationen im Computer. Das Problem, Eins und Eins zusammenzuzählen

Die Befehlssätze der meisten heute lieferbaren Mikrocomputer sind auf die Addition und Subtraktion ganzzahliger Binärzahlen bestimmter Länge (max. 16bit) beschränkt. Alle übrigen mathematischen Operationen, sei es andere Funktionen, größere Zahlen oder Zahlen in Potenzschreibweise (="Gleitkomma–Arithmetik"), muß man sich selbst auf der Basis von 1 - und 2Byte–Addition und –Subtraktion in Form eines Programms erstellen bzw. entsprechende von Anbieter gelieferte Standardprogramme benutzen.

Es ist daher erforderlich, sich Grundkenntnisse darüber anzueigenen, wie Rechnungen im Dualsystem durchgeführt werden:

o Die ADDITION

$0 + 0 = 0$
$1 + 0 = 1$
$0 + 1 = 1$
$1 + 1 = 0$ mit Überlauf auf die nächst höhere Dualstelle

Beispiel: 3719
\+ 38
\+ 11
3768_{10}

1110 1000 0111
\+ 10 0110
\+ 1011
$1110\ 1011\ 1000_2$

o DIE SUBTRAKTION

$0 - 0 = 0$
$1 - 0 = 1$
$0 - 1 = 1$ mit Unterlauf der nächst höheren Dualstelle
$1 - 1 = 0$

Beispiel: 3719
\- 38
3681_{10}

1110 1000 0111
\- 10 0110
$1110\ 0110\ 0001_2$

o Die MULTIPLIKATION

$0 \times 0 = 0$
$1 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 1 = 1$

Die Multiplikation zweier Dualzahlen geschieht durch die Addition der geeigneten stellenversetzten Produkte des Mulitplikators mit dem Multiplikanden. Da als Ziffernwerte nur 0 und 1 auftreten, entfällt der im Dezimalsystem notwendige Aufwand (kleines Einmaleins) völlig.

Beispiel: 100 x 27
2700_{10}

1100100 x 11011
1100100
1100100
0
1100100
1100100
101010001100_2

o Die DIVISION

Bei der Division zweier Dualzahlen ergeben sich die Dualstellen des Quotienten als diejenigen Vielfachen des Divisors, die geeignet stellenversetzt im Divisor bzw. in den jeweils verbleibenden Resten (ohne Vorzeichenwechsel) enthalten sind. Da als Ziffernwerte nur 0 und 1 vorkommen, ist lediglich darauf zu achten, daß der Wert des Divisors den jeweiligen Divisionsrest nicht übersteigt.

```
Beispiel:   2700 : 100 = 27₁₀
            200
            ---
             700
             700
             ---
               0

101010001100 : 1100100 = 11011₂
1100100
--------
10001001
 1100100
--------
 10010110
  1100100
---------
  1100100
  1100100
---------
        0
```

o Die Bildung des Komplements

Wie bereits gezeigt, ist die Multiplikation zweier Dualzahlen lediglich eine Addition mit Stellenverschiebung. Wenn es gelingt, die Subtraktion auf eine Addition zurückzuführen, dann ist auch die Division zweier Dualzahlen wegen des Vorganges der stellenversetzten Subtraktion mittels einer Addition auszuführen.

Eine derartige Rückführung ist möglich durch Bildung des sogenannten Zweier-Komplements.

Die Vorgehensweise ist wie folgt:

– Die negative (zu subtrahierende) Dualzahl ist positiv aufzuschreiben.

– Der Subtrahend ist, soweit nötig, auf die Stellenzahl des Minuenden durch Nullen aufzufüllen.

– Sämtliche Dualstellen der Zahl sind umzuwandeln, d.h. aus 0 wird 1 bzw. aus 1 wird 0. Dieser Vorgang bedeutet Bildung des Komplements.

– Das Komplement wird nun zum Minuenden addiert. Der sich auf der am weitesten links stehenden Stelle ergebende Überlauf ist auf das Ergebnis in der am weitesten rechts stehenden Dualstelle zu addieren. Die sich auf diese Weise ergebende Dualzahl ist identisch mit derjenigen, welche im Wege der normalen Subtraktion errechnet wird.

```
Beispiel:   3719  dem entspricht    3719
           − 38                    + 62
           -----                   -----
           3681₁₀                  3781
                                   −100
                                   -----
                                   3681₁₀
```

$$
\begin{array}{lrl}
\text{Minuend} & 111010000111 & - \; 000000100110 \;\; \text{Subtrahend} \\
 & +\; 111111011001 & \quad 111111011001 \;\; \text{Komplement} \\
\hline
 & 1111001100000 & \\
 & \downarrow & \\
+ & \longrightarrow 1 & \\
\hline
 & 111001100001_2 &
\end{array}
$$

o Die Darstellung von Brüchen

Zur Darstellung von Bruchzahlen werden im Dezimalsystem die Stellenwerte

$$\frac{1}{10^1}, \frac{1}{10^2}, \frac{1}{10^3}, \ldots$$

eingeführt und die zugehörigen aktuellen Stelleninhalte rechts neben dem Komma aufgeschrieben, das nach der Einerstelle gesetzt wird. Entsprechend stehen im Dualsystem nach dem Komma rechts von der Einerstelle die Stelleninhalte, die zu den Stellenwerten

$$\frac{1}{2^1}, \frac{1}{2^2}, \frac{1}{2^3} \ldots,$$

gehören.

Beispiel: Der Dualbruch $0{,}1011_2$ bedeutet als Dezimalbruch:

$$\frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} = 0{,}6875_{10}$$

Die Divison von Dualzahlen kann somit über die Einerstelle des Quotienten hinaus fortgesetzt werden.

Alle übrigen Rechenoperationen bzw. Funktionen werden auf die in der mathematischen Standardliteratur beschriebenen **Reihenentwicklungen** zurückgeführt, einfach in **tabellarischer Form** berechnet oder aber aufgrund spezieller, für Rechnungen im Dualsystem eigens entwickelter **Binär-Algorithmen** abgearbeitet.

1.9. Fest - und Gleitkommaarithmetik

Rechnen mit Dualzahlen ist zwar im Computer die schnellste Arbeitsweise, kann jedoch durch Konvertierungs- - bzw. Dekonvertierungsaufwand recht umständlich und langsam werden. Im Falle rechenintensiver Aufgabenstellungen ist jedoch Dualarithmetik zwingend. Dies gilt sowohl aus Zeitgründen wie aus Gründen der Kommaverwaltung. Die Darstellung von Dualzahlen kann in

– Festkomma -Arithmetik oder

– Gleitkomma–Arithmetik

vorgenommen werden.

a) Festkomma–Arithmetik

Solange beim Rechnen mit Dezimal - oder Dualzahlen ungebrochene, ganze Zahlen in Betracht kommen, bleibt eine Kommastellung unberücksichtigt. Auch bei der Verarbeitung von Dezimal– oder Dualbrüchen bestehen keine besonderen zusätzlichen Probleme, solange es sich um Operationsvorgänge wie Übertragen, Addieren, Subtrahieren oder Multiplizieren handelt. Die Stellung des Kommas in der Ergebniszahl wird ebenso wie die Länge des Bruches und das Runden vom Anwender durch Programmdefinitionen bestimmt. Dies ist deshalb möglich, weil z.B. beim kommerziellen Rechnen die Kommastellung durch Pfennigbeträge bestimmt ist und in der Programmierung bei der Vorbereitung zur Datenausgabe relativ einfach berücksichtigt werden kann. Die Speicherung und Rechenfolge in der CPU läuft also ohne interne Kommaverwaltung ab.

Diese Art der Speicherung und Verarbeitung von Dualzahlen ohne Kommaberücksichtigung heißt **Festkomma-Arithmetik.**

Die Feldlänge für duale Festkommazahlen ist variabel und muß vom Anwender durch Anweisungen eigens festgelegt werden.

b) Gleitkomma–Arithmetik

Bei rechenintensiven Programmen, bei denen zum einen die aktuellen Feldlängen der Ergebnisoperanden nicht im voraus abgeschätzt werden können, zum andern vor allem aber die Kommastellung häufig wechselt, muß die Gleitkomma-Arithmetik zur Darstellung von Dualzahlen angewendet werden. Die Gleitkommadarstellung gestattet, relativ große Zahlenbereiche mit einheitlichem Format und hoher Genauigkeit abzubilden. Zur Darstellung des Prinzips, auf dem Gleitkommazahlen aufbauen, soll das Zehnerzahlensystem dienen.

Die Umwandlung einer gebrochenen Zahl in eine Gleitkommazahl geschieht durch eine Zifferntrennung von der Größenordnung der Bruchzahl. Die Gleitkommazahl besteht danach aus zwei verbundenen Werten:

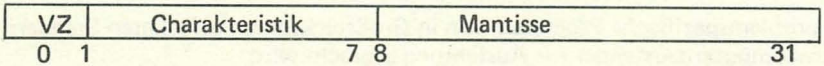
– Mantisse
– Exponent

$$+\ 386 = 0{,}386 \times 10^{3} \leftarrow$$
$$-\ 0{,}386 = 0{,}386 \times 10^{0} \leftarrow \text{Exponent}$$
$$+\ 0{,}0\ 00\ 0\ 386 = 0{,}386 \times 10^{4} \leftarrow$$

↑

Mantisse

Die **Mantisse** gibt denjenigen Ziffernteil einer Gleitkommazahl an, vor dem links vor der höchsten Ziffer das Komma angenommen werden muß, wenn, wie im Beispiel, die höchste Ziffer ungleich Null ist, so gilt die Gleitkommazahl als "normalisiert".
Der **Exponent** wird als Potenz zur Basiszahl angegeben und berücksichtigt die tatsächliche Kommastellung der Zahl. Gleitkommazahlen werden in Formaten fester Länge gespeichert:



- Das erste Bit (Position 0) einer Gleitkommazahl gibt das Vorzeichen (VZ = $\pm$) der Zahl an.
- Die Bitposition 1 – 7 enthalten die sogenannte Charakteristik. Dual dargestellt, kann in dieser Feldlänge die Zahl 0 – 127 dargestellt werden. Zur Ausschaltung einer weiteren Vorzeichenangabe für Exponenten wird dem Exponenten "0" die Charakteristik 64 zugeordnet; dadurch sind Angaben für Hochzahlen von – 64 bis + 63 möglich. Die Größe der aktuellen Charakteristik ergibt sich jeweils aus der Addition von 64 zum Exponenten.
- Die verbleibende Wortlänge dient der Aufnahme der Mantisse.

Eigene Befehle für Gleitkomma–Arithmetik können in Mikroprozessoren wegen des damit verbundenen hohen Hardware–Aufwandes nicht implementiert werden; man löst daher Gleitkommaprobleme mit Unterprogrammen, die als Standard–Anwendersoftware verfügbar sind.

2. Die Bestandteile von Computersystemen – Mega, Mini oder Mikro

2.1. Computerstruktur

Nach unserer theoretischen Vorarbeit in Abschnitt 1 über Begriffe der Computertechnik und Zahlen wollen wir jetzt die Funktionsweise des Computers selbst angreifen.

Im Vordergrund steht wieder die Technik des Mikrocomputers, aber es soll noch einmal daran erinnert werden, daß auch das im Folgenden gesagte sinngemäß für andere Rechenanlagen gilt, da der Mikrocomputer nur ein "Spezialfall" einer "Allgemeinen" Maschine (="General Purpose Computer") ist.

Wesentlichstes Kennzeichen eines Mikrocomputers gegenüber einer konventionellen Realisierung ist die Tatsache, daß dort

- die Gerätefunktion durch ein Maschinenprogramm erfüllt wird*, das in binärer Form in Halbleiterspeicher-Bausteinen abgelegt ist,
- dieses Maschinenprogramm sequentiell (d.h. ein Maschinenbefehl nach dem anderen) ausgeführt wird und dadurch die Gerätefunktion erfüllt wird.
- das problemspezifische Programm von in Großstückzahlen gefertigten Standard Mikrocomputerbausteinen zur Ausführung gebracht wird.

Aus diesem Grund ist trotz der bereits allgemein hohen Vorarbeitungsgeschwindigkeiten der Durchsatz ("Throughput") des verwendeten Mikroprozessors häufig ein wichtiges Kriterium für die Anwendbarkeit der Mikrocomputertechnik.
Insbesondere gilt dies für Fälle, bei denen das Gerät innerhalb einer gewissen Minimalzeit auf bestimmte Situationen reagieren muß (z.B. auf den Stromausfall bei einer Maschinensteuerung als Schutz gegen Beschädigung von Werkstück oder Werkzeug).

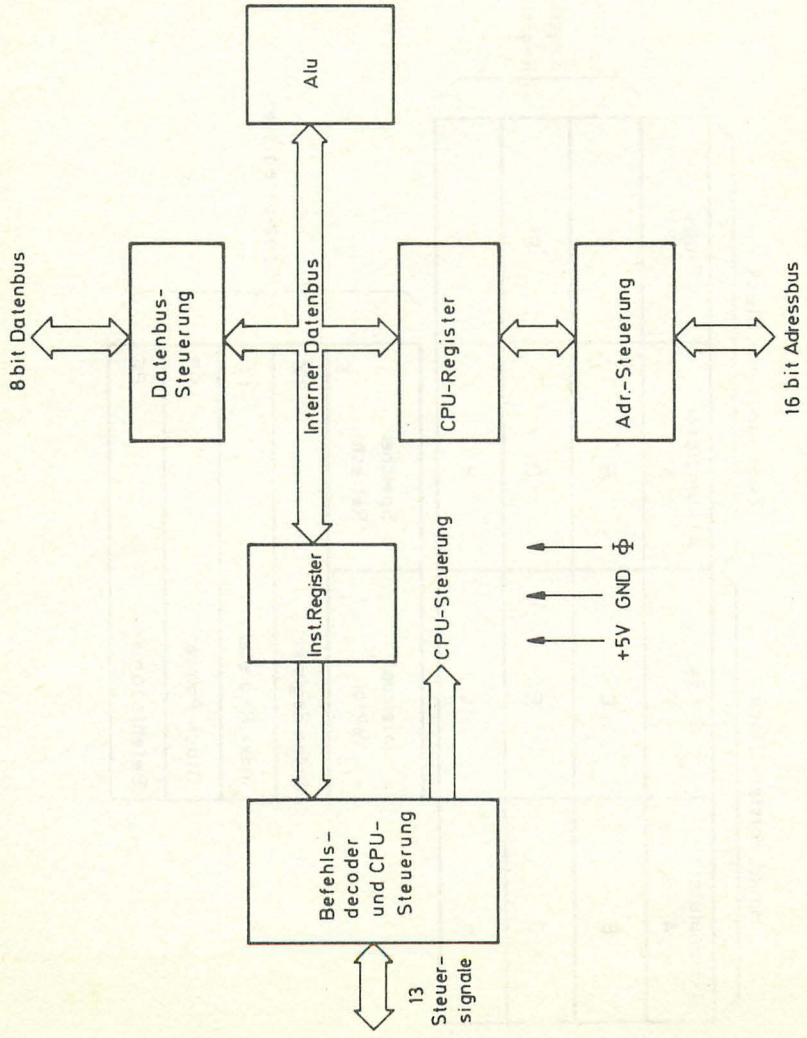
* Bemerkung
Hieraus erklärt sich die Grenze der Mikrocomputertechnik:
Da die einzelnen Aktionen des Gerätes durch aufeinanderfolgende Programmschritte bewirkt werden, die selbst als Geschwindigkeitsgrenze die maximale Geschwindigkeit der verwendeten Halbleitertechnologie haben, ist evtl. mit einem einzigen, sequentiell arbeitenden Mikroprozessor die erforderliche Systemreaktionszeit nicht zu erreichen.

2.2. Zentralprozessor - "Gehirn" des Computers

Die Zentrale, die einen Großteil der genannten Aktivitäten übernimmt, ist der Mikroprozessor.
In ihm ist die "technische Intelligenz" konzentriert.

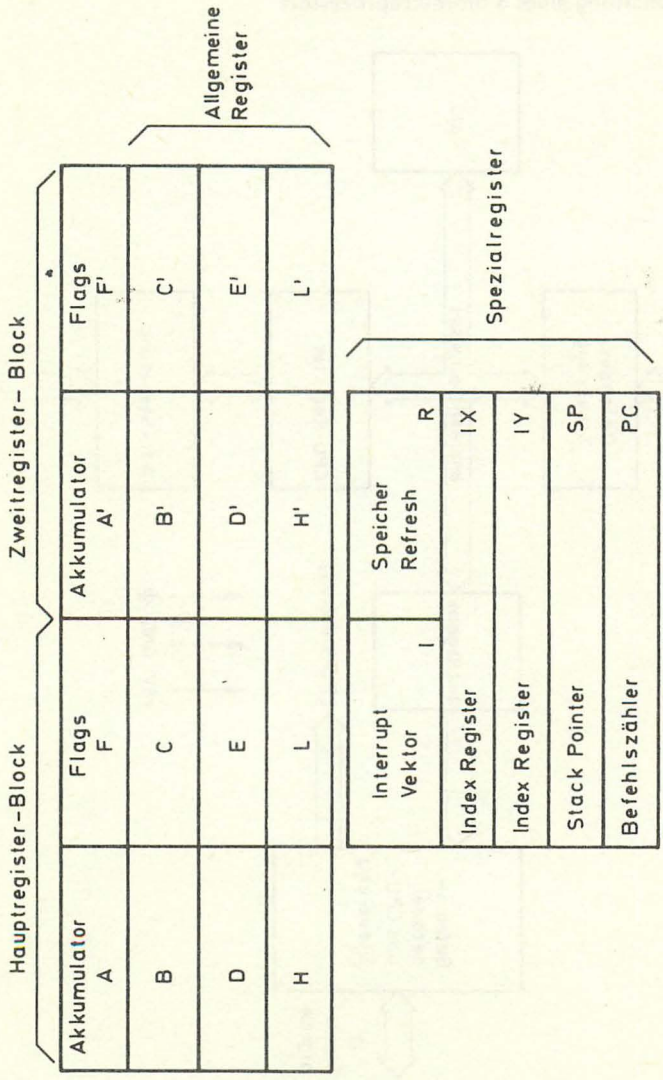
Hierzu braucht man die in Bild 4 gezeigten Grundeinheiten (als Beispiel dient hier das Blockbild des Mikroprozessors Z80–CPU):

Bild 4: Blockschaltung eines 8 bit-Mikroprozessors



– Registerblock: Er besteht aus einer Anzahl Speicherzellen, in denen Zwischenergebnisse und sonstige Informationen abgelegt sind, auf die der übrige Prozessor in möglichst kurzer Zeit zugreifen können muß.
Auf einen Teil dieser Register kann das Anwenderprogramm zugreifen, andere sind ausschließlich für interne Zwecke reserviert (Bild 5).

Bild 5: Schematische Darstellung des Registerblocks



– Rechenwerk: (auch arithmetisch/logische Einheit = arithmetic/logic unit = ALU).
Es führt arithmetische (Addition, Subtraktion) und logische (logisches UND, ODER, EXCLUSIV ODER) Operationen durch, auf die sämtliche Rechnungen und Entscheidungen des Prozessors zurückgeführt werden können. Das oder die Rechenregister, worin arithmetisch/logische Operationen durchgeführt werden, heißen Akkumulator oder Akku.
Da man sich darüber streiten kann, ob sie zum Registerblock oder zum Rechenwerk zu zählen sind, bezeichnet man häufig Register und Rechenwerk zusammen als "RALU" (="Register and Arithmetic/Logic Unit").

– Steuerwerk: Das ist die "Schaltzentrale" des Zentralprozessors. Hier werden mit Hilfe einer als ROM im Zentralprozessor integrierten Firmware, dem sogenannten "Mikroprogramm"*, die vom Prozessor eingelesenen Maschinenbefehle "decodiert", also in die eigentlichen Funktionen umgesetzt.

Dazu gehört neben dem "Weichenstellen" für den prozessorinternen Datentransport das Bestimmen interner Zeitabläufe, Prüfung vorliegender Bedingungen, die in einem eigenen Register gespeichert sind, Festlegung der momentan vom Rechenwerk auszuführenden Funktion und das Fortschreiben eines Sonderregisters (="Befehlszähler" = "Program Counter"), in dem zu jedem Zeitpunkt die Adresse des als nächstes auszuführenden Befehls steht.
Für die oben erwähnte Prüfung von Bedingungen benützt man übrigens sogenannte "Bedingungsbits" oder "Flags", die in einem oder mehreren Registern abgespeichert sind.
Diese Bedingungsbits werden von gewissen Befehlen beeinflußt, d.h. je nach Resultat des entsprechenden Befehls auf logisch Null oder Eins gesetzt.
So gibt es z.B. ein "Zero-Bit", das immer dann 1 (="wahr") wird, wenn das Ergebnis der letzten Operation gleich Null war, bzw. zu Ø (="falsch") wird, wenn das Ergebnis der letzten Operation ungleich Null war.
Der Zustand dieser Bedingungsbits gibt also Auskunft über die zuvor im Programm aufgetretenen Resultate und ermöglicht dadurch auch die Ausführung unterschiedlicher Aktivitäten je nach vorliegendem Ergebnis (="Verzweigungen").

Schließlich benötigt die CPU noch folgende zwei Hilfseinrichtungen:

–Den Taktgenerator, der alle Vorgänge im Computersystem synchronisiert.
Alle käuflichen Mikrocomputer sind "Synchron-Maschinen", d.h. die nächstfolgende Aktivität wird auf Gatterebene nicht durch das Ende der vorhergehenden eingeleitet, sondern durch ein vom Systemtakt festgelegtes Zeitraster.

* bitte erinnern Sie sich an das in Pkt. 1.5. Gesagte!

– Den Systembus, der aus Adreßbus, Datenbus und Steuerbus besteht. Auf ihm werden Adreß– und Steuerinformationen auf Adreß– und Steuerbus und die Maschinencodes der auszuführenden Befehle und die zugehörigen Daten auf dem Datenbus zwischen dem Mikroprozessor und den übrigen Mikrocomputerbausteinen übertragen.
Diese Busse sind unidirektional, wenn nur ein Baustein als Sender in Frage kommt, (wie das z.B. beim Adreßbus der Fall ist, bei dem im allgemeinen ausschließlich die CPU die gültige Adresse ausgibt) oder bidirektional, wie das beim Datenbus der Fall ist, wo die CPU z.B. Informationen von einem Schreib/Lesespeicher liest oder aber sie ihm Daten übergibt. Verbindet man eine solche "intelligente" Zentraleinheit entsprechend Bild 2 mit den für die Ein/Ausgabe–, Decodier – und Treiber–Bausteine), so kann der auf diese Weise entstandene Computer eine Gesamtheit von Aktivitäten ausführen, die als "Anwenderprogramm" vorgegeben sind und die die Gerätefunktion verwirklicht.

2.3. Vom Befehl zum Anwenderprogramm;

Die untersten Elemente eines Programms sind die Maschinenbefehle (= "Maschinen· instruktionen"), die der Zentralprozessor direkt in sein Befehlsregister (="Instruction Register") hereinholen, im Steuerwerk decodieren und schließlich ausführen kann. Die Gesamtheit dieser Maschinenbefehle bezeichnet man als Befehlsvorrat (="Instruction Set") eines Zentralprozessors..
Umfang des Befehlsvorrats und Leistungsfähigkeit der einzelnen Befehle haben direkten Einfluß auf die Leistungsfähigkeit des gesamten Computersystems (Speicherplatzausnutzung und Verarbeitungsgeschwindigkeit).
Durch diese Befehle werden elementare Arbeitsschritte angesprochen, die durch das Steuerwerk der Zentraleinheit zur Ausführung gebracht werden. Dazu greift das Steuerwerk auf das aus einzelnen Instruktionen bestehende Programm im Speicher zu und ruft über die im Befehlszähler stehende Adresse jeweils die an der Reihe befindliche Instruktion ab.

Jede Instruktion bewirkt eine bestimmte Operation. Die Gesamtheit aller Instruktionen, die von einem bestimmten Prozessor abgewickelt werden können, wird als Instruktionsvorrat oder Befehlsvorrat bezeichnet. Der Instruktionsvorrat kann bei einzelnen Computertypen außerordentlich unterschiedlich sein.
Jede Instruktion besteht aus mindestens zwei verschiedenen Elementen:

- o dem Operationsteil (="Opcode")
- o dem Adreßteil

Format:

Operationsteil	Adreßteil

Im Operationsteil ist im Binärcode diejenige Operation verschlüsselt, die bei Ausführung der Instruktion angesprochen werden soll ("was muß getan werden").
Über den Adreßteil erfolgt die Bezugnahme auf diejenigen Operanden, die von der Operation betroffen werden sollen ("worauf bezieht sich diese Aktion").

Im allgemeinen erfolgt die Bezugnahme über Adressen, durch die der Platz im Speicher lokalisiert wird, auf dem die betreffenden Operanden zu finden sind. Daher rührt der Name "Adreßteil". Während einer Instruktionsausführung werden die Operanden über diese Adressen aufgesucht.

Bei einigen Instruktionen stehen im Adreßteil anstelle der Adressen die Operanden selbst. Derartige Operanden werden demzufolge als Direktoperanden bezeichnet ("Immediate operation").

Abhängig von der Anzahl der im Adreßteil einer Instruktion erscheinenden Operandenadressen bzw. Direktoperanden werden diese Instruktionen als Einadreß– bzw. Mehradreßinstruktionen (–befehle) und die sie repräsentierenden Zentraleinheiten als Einadreß– bzw. Mehradreßmaschinen bezeichnet.

Zur Zeit gängige MOS–Mikroprozessoren sind sämtlich Einadreß–Maschinen.

Zwischen den internen CPU–Registern sind allerdings auch meist Mehr–Adreß–Datenübertragungen möglich.

Der Adreßteil von Einadreßinstruktionen nimmt, wie der Name besagt, eine einzige Adresse auf. Diese Adresse gibt entweder an WOHER (d.h. von welcher Speicheradresse) ein Operand vom Steuerwerk abgerufen oder WOHIN (d.h. auf welche Speicheradresse) ein Operand bzw. ein Operationsergebnis zurückgeschrieben werden soll. Dabei erhebt sich die grundsätzliche Frage nach der zweiten Adresse eines jeden Operanden. Wenn das Steuerwerk einen Operanden beispielsweise von dem Speicherplatz mit der Adresse 805 abrufen soll, muß im gleichen Vorgang sichergestellt werden können, auf welchen anderen Platz der Inhalt der genannten Adresse gestellt werden soll. Ein derartiger Adreßverweis fehlt aber im Adreßteil der Instruktion. Sie ist auch überflüssig, weil bei jeder Operation ein bestimmtes Register des Zentralprozessors beteiligt ist, in das hinein Operanden geladen bzw. aus dem Operanden oder Operationsergebnisse abgerufen werden. Der Befehlsvorrat von Einadreßmaschinen muß aus diesem Grunde auch Befehle zum Laden und Entladen von Registern enthalten.

Beispiel:

Es sind die beiden 8bit–Binär–Zahlen zu addieren, die sich in Register B bzw. auf dem Speicherplatz mit der Adresse 6F33 befinden. Die Summe soll auf den Speicherplatz des zweiten Operanden, d.h. nach 6F33 gebracht werden.

Die hierzu erforderliche Instruktionsfolge könnte wie folgt aussehen:

LD A, (6F33H)	1. Instruktion
ADD B	2. Instruktion
LD (6F33H), A	3. Instruktion

Durch die erste Instruktion wird das Laden des Akkumulators mit der auf dem Speicherplatz mit der Adresse 6F33 stehenden Zahl bewirkt. Die zweite Instruktion hat zur Folge, daß zum Akkumulatorinhalt die Zahl addiert wird, die im Register B steht. Die dritte und letzte Instruktion bewirkt schließlich das Rückspeichern des Akkumulators; sein Inhalt wird nach 6F33 gespeichert. Beim Datentransfer bleiben die Inhalte vom Akkumulator, Register und Speicherstellen grundsätzlich erhalten.

Bitte machen Sie sich nochmals klar, daß wir es hier mit **Einadreß-Befehlen** zu tun hatten, da nur 1 Speicheradresse in 1 Befehl vorkam. Vor dieser Klassifizierung zu unterscheiden ist die Aussage, ob es sich bei einem Befehl um einen **Ein-Byte-** oder **Mehr-Byte-Befehl** handelt.

Damit ist ganz einfach die Länge der Binärkombination gemeint, durch die ein Befehl dargestellt ist.
Beim Durchsehen des Befehlsatzes eines Prozessors wird der Grund für die unterschiedliche Länge der Befehle klar.
Zur Erklärung soll hier wieder die 8bit–Maschine Z80–CPU herangezogen werden.
Bei ihr haben Operanden im Allgemeinen die Wortlänge 8bit, Adressen dagegen 16bit (s. nächster Abschnitt).

Wenn wir zunächst annehmen, daß die Länge des Operationsteils (="Op–Codes") konstant und gleich 8bit ist, dann ist die Gesamtlänge des Befehls, der aus Operationsteil und Adreßteil besteht, natürlich unterschiedlich, je nach dem, ob im Adreßteil gar nichts steht (wie bei Befehl NOP = No Operation = Leerbefehl) oder ein 8bit–Operand.

1. Byte
(wie beim Befehl ADD 07H) oder aber eine 16bit-Adresse
1. Byte 2. Byte
(wie beim Befehl JP (47 11 H).
1. Byte 2. Byte 3. Byte

Zur Beachtung: Der Buchstabe "H" kennzeichnet lediglich das Zahlensystem Hexadezimal und tritt im eigentlichen Maschinenprogramm nicht in Erscheinung, da dort die einzig mögliche Zahlendarstellung sowieso die binäre ist.

Hinzu kommt, daß für Hochleistungs–Mikroprozessoren wie der Z80–CPU die Anzahl der durch den 8bit–Opcode darstellbaren Instruktionsvorrat von $2^8 = 256$ Befehlen nicht ausreicht und daher teilweise 2–Byte–Opcodes benützt werden.

Dadurch kommen bis zu 4 Byte lange Befehle zustande.
Aus solchen Befehlen – seien sie je nach ihrer Funktion ein, zwei, drei oder vier Bytes lang – wird nun das ganze Anwenderprogramm aufgebaut, in dem der Maschinencode dieser Befehle nacheinander in (meist) lückenloser Folge in den Programmspeicher gebracht werden; wie dieses "Einlesen" geschieht, werden wir später noch detailliert behandeln.

Aus dem Programmspeicher werden die Befehle bei der Programmausführung (die einfach durch Setzen des Befehlszählers auf die Speicherzellennummer geschieht, in der der erste auszuführende Befehl steht) in die CPU gelesen und von ihr verarbeitet.
Es soll hier nochmals betont werden, daß die Verarbeitung rein sequentiell geschieht, also der Prozessor nie 2 Aktivitäten gleichzeitig durchführen kann.

Anders ist das bei festverdrahteter Logik, wo man mehrere Funktionalblöcke aufbauen kann, die **gleichzeitig** arbeiten, was natürlich zeitsparender ist.

Daher sind Hardware-Realisierungen bei extrem zeitkritischen Reaktions-Problemen die einzig gangbare Lösung – gleichgültig ob man dabei dem Rechner mit Zusatzhardware Arbeit abnimmt oder ein ganzes Hardware-System spezifisch "schneidert".

Ein anderer Weg zur Realisierung "superschneller" Anlagen ist die Verwendung von Prozessoren in bipolarer Technik (z.B. Schottky-TTL oder ECL) oder gar die Aufteilung der Gesamtaufgabe auf mehrere Mikroprozessoren (="Multi-Mikroprozessor-System").

Da aber moderne, heute verfügbare nMOS-Mikroprozessoren wie der Z80A-CPU aufgrund ihrer Architektur und ihrer hohen Taktfrequenz (4 MHz!!) bereits sehr hohe Durchsätze erlauben, gibt es nur sehr wenig Probleme, bei denen man aus Geschwindigkeitsgründen auf Bipolare Mikrocomputer - Systeme übergehen muß.

2.4. Adressierung

Bisher wurde schon öfter auf die Tatsache Bezug genommen, daß in Computern Befehle und Daten im Speicher abgelegt sind und über "Adressen" aufgefunden (also aus der Speicherstelle ausgelesen bzw. in die Speicherstelle eingelesen) werden können.
Eine Adresse ist dabei nichts weiter als ein Suchbegriff; die einzelnen Speicherstellen sind "durchnummeriert" und die Elektronik ("Hardware") ist so eingerichtet, daß beim Anlegen einer binären Zahl (=bit-Kombination) an die Adreßleitungen des Speichers an seinen Datenleitungen die Informationen erscheinen, die in der Speicherstelle mit der entsprechenden Adresse momentan stehen (Bild 5).

Es muß nun noch konkretisiert werden, wie diese Speicherstellen aufgebaut sind und wie das Ansprechen (="Adressieren") vor sich geht.

In Abschnitt 2.7. werden die Realisierungsmöglichkeiten von Speicherorganisation besprochen; zunächst müssen wir jedoch Speicherorganisation und Adressierungsverfahren klären.
Der Speicher eines Mikrocomputers ist "wortweise aufgebaut", oder, wie man in der Computertechnik sagt, "wortweise organisiert". Das heißt, daß unter dem Suchbegriff (=1 Adresse) genau soviele Bits abgespeichert sind, wie der Zentralprozessor "auf einmal" (also parallel) verarbeiten kann.
In 8bit-Mikrocomputersystemen verwendet man also Speicher, in denen pro Adresse 1 Byte (=8bit) angesprochen wird, bei 16bit-Mikrocomputern steht in jeder Speicherzelle 1 Sechzehn-Bit-Wort. Der Adressierumfang (also die maximal ansprechbare Anzahl von Speicherzellen) ist durch die Anzahl der Adreßleitungen gegeben. 8 und 16bit Mikrocomputersysteme verfügen meist über 16 Adreßbits und dadurch über einen Adressierumfang von 2^{16} = 64k = 65 536 Worten. Dieser Adressierumfang kann durch gewisse Hard- und Software-Vorkehrungen (=sogenannte PAGING-Technik) erweitert werden.

Das Auffinden eines Speicherplatzes kann auf sehr unterschiedliche Weise ("Adressierweise") erfolgen. Für den "Computer-Anfänger" ist es nützlich, sich die Funktion dieser Adressierweisen an folgendem leichtverständlichen Analog einer Kartei klarzumachen.
Jede Karte entspricht dabei einem **Speicherplatz**, die Aufschrift auf der Karte ist der **Speicherinhalt**. Dieser kann entweder aus Daten bestehen (OFH) oder aus **Anweisungen** (z.B. "Nimm nächste Karte!"). Jede Karte ist, von außen erkennbar, mit einem Schildchen versehen, auf dem seine Adresse vermerkt ist.

Wir wollen nun eine Reihe Adressierweisen durchgehen, wobei jeweils zu Anfang der analoge Vorgang beim Kartensuchen genannt wird:

a) Direkte Adressierung:

(=die Nummer der gesuchten Karte ist bekannt. Sie wird aufgrund der Aufschrift auf dem Adreß-Schildchen herausgesucht).
Im Maschinenbefehl steht expliziert die volle direkte (="absolute") Adresse.
Beispiel: JP 4711H
bewirkt einen Sprung (="Jump") zur hexadezimal angegebenen Speicheradresse 4711.

b) Relative Adressierung

(=es wird nicht die Karte gesucht, deren Adresse bekannt ist, sondern eine solche, die sich in einer ebenfalls gegebenen Distanz von x Karten vor oder nach der bekannten Adresse befindet. Sie wird herausgesucht, indem man die Karte mit der bekannten Adresse aufsucht, aber nicht herauszieht, sondern die Distanz x vor- oder zurückgeht und die dann aufgefundene Karte liest).

Die relative Adressierung wird bei Mikrocomputersystemen zur Einsparung von Programmspeicherplatz durch Verwendung relativer Sprünge verwendet.

Dahinter steckt folgender Vorgang:
Bei einem Sprung, dessen Ziel irgendwo im 64k großen Speicher liegt, muß natürlich eine volle 16bit-Adresse mit in dem Sprungbefehl angegeben werden, was jedesmal 2 Byte Programmspeicherplatz kostet.
Da die meisten Sprünge aber nur über eine relativ kurze Distanz (z.B. über ± 128 Speicherstellen) nötig sind, würde eigentlich eine 8bit ($2^8 = 256$)-Distanzadresse (="Displacement") genügen, die nur 1 Byte Programmspeicherplatz erfordert.
Man benutzt daher solche relativen Sprünge, bei denen im Operandenteil lediglich ein Vorzeichen (1 Bit = Plus oder Minus) und eine 7bit-Distanzadresse steht.
Der Sprung wird dann einfach dadurch ausgeführt, daß dieses "Displacement" zum momentanen Stand des Befehlszählers vorzeichenrichtig addiert wird und der Inhalt der so aufgefundenen Speicherzelle als nächstauszuführender Befehl interpretiert wird.

Beispiel: JR -5

Würde einen Sprung vom momentanen Stand des Befehlszählers um 5 Speicherstellen zurück veranlassen (= "Jump Relative").

c) Indizierte Adressierung
(= Vor Aufsuchen der angegebenen Adresse wird zum Wert dieser Adresse eine Zahl dazu addiert, die man sich außerhalb der Kartei notiert hat, nämlich im Indexregister).
Dabei wird der Inhalt des sogenannten I n d e x r e g i s t e r s zur gegebenen Adresse addiert.
Indexregister sind Teile des Registerblocks des Zentralprozessors.
Der Programmierer hat bei Indexregistern für das Laden der Registerinhalte und ggf. für deren Veränderung im Programmlauf zu sorgen.
Indexregister werden vorteilhaft dann verwendet, wenn es sich – wie bei mehrfachem Durchlaufen von Programmschleifen – anbietet, z.B. Basisadressen durch Addition einer bestimmten Zahl zu verändern (Adreßmodifikation).
Ebenso ist eine Adreßbildung durch Indizierung dann von großem Wert, wenn man den Registerinhalt fortlaufend byteweise oder in Sprüngen von mehreren Bytes erhöht, um ein Datenfeld (Matrix) nach bestimmten Komponenten zu durchsuchen oder bei Mehrfachprogrammverzweigungen (= "Sprungtabellen").
Beispiel: Der Befehl

INC (IX+5H)

würde die Speicherstelle aufsuchen, die durch den Inhalt des Indexregisters plus 5 gegeben ist und deren Inhalt inkrementieren (= um 1 erhöhen).

d) Indirekte Adressierung
(= man sucht die Karte mit der angegebenen Adresse, faßt aber dann den Inhalt dieser Karte als die eigentliche Adresse auf und sucht nun diese. Erst der Inhalt der zweiten aufgefundenen Karte ist dann die gesuchte Information).

Bei der indirekten Adressierung muß man zwei Varianten unterscheiden:

- die speicherindirekte Adressierung. Dabei wird der Inhalt der im Adreßteil des Befehls angegebenen Speicherstelle seinerseits als Adresse interpretiert und dann der Inhalt jener Speicherstelle gelesen.
  Nachteil des Verfahrens ist, daß hier bis zum Auffinden der gesuchten Information zwei volle Arbeitsspeicherzugriffe nötig sind, die bekanntlich wesentlich mehr Zeit in Anspruch nehmen als Zugriffe auf interne Register des Zentralprozessors.
  Aus diesem Grund verwendet man in der Mikrocomputertechnik meist
- die registerindirekte Adressierung. Bei ihr wird im Adreßteil des Befehls lediglich eine Registerbezeichnung angegeben und dann sofort **die** Speicherstelle gelesen, deren **Adresse** in dem im Befehl angegebenen Register bzw. Registerpaar steht.

Die indirekte Adressierung bietet eine ähnliche Flexibilität in der Programmerstellung wie die indizierte Adressierung, wobei lediglich das Verfahren der Ermittlung der effektiven Adresse unterschiedlich ist: Während bei der indizierten Adressierweise die effektive Adresse durch Addition eines Indexregisters ermittelt wurde, wird bei der indirekten Adressierung der Wert des Adreßregisters unverändert als effektive Adresse interpretiert.

Beispiel: Der Befehl

```
LD A, (DE)
```

würde den momentanen Inhalt des Registerpaares DE als Adresse auffassen, den Inhalt der so adressierten Speicherzelle in den Zentralprozessor lesen und dort in den Akkumulator (= "A") transferieren.

e) Die unmittelbare Adressierung haben wir schon im Abschnitt 2.3. "als Dierektoperation" (bitte nicht mit direkter Adressierung verwechseln!) kennengelernt. Darunter versteht man nichts anderes als das Behandeln einer Konstanten im Adreßteil eines Befehls. Zum Beispiel würde der Befehl

```
LD HL, 4711H
```

die Adresse (bzw. die Konstante) 4711 (hexadezimal) in das Registerpaar HL schreiben.

f) Registeradressierung
Auch das Nennen eines Registernamens im Adreßteil eines Befehls ist eine Form der Adressierung, nur wird dabei eben nicht eine Zelle des (externen) Arbeitsspeichers angesprochen, sondern der Inhalt des genannten Registers (oder eines Registerpaares).

Beispiel: der Befehl

```
LD C, L
```

würde die Register C und L ansprechen, wobei der Inhalt von Register L ins Register C kopiert würde.

g) Implizierte Adressierung

In manchen Befehlen ist bereits durch den Typ (genauer: durch den OP-Code) des Befehls das Ansprechen eines bestimmten Registers festgelegt. So wird bei Additionen bei Einadressmaschinen (was Mikrocomputer sind) bei arithmetischen Operationen als zweiter Operand **und** als Register, in das das Ergebnis abgelegt wird, immer stillschweigend der Akkumulator angenommen.

Beispiel: Der Befehl

ADD B

nimmt den Inhalt des Akkumulators ("Register A"), nimmt den Inhalt des Registers B, addiert die Inhalte und legt sie wieder im Akkumulator ab.

Der Befehl

ADD A

dagegen würde den Akkuinhalt zu sich selbst addieren (also mit 2 multiplizieren).

h) Bit-Adressierung

Da der Speicher wortweise organisiert ist, in vielen Anwendungen jedoch der Zugriff auf das einzelne Bit erforderlich ist (z.B. in Maschinensteuerungen, wo einzelne Bits entsprechend einzelnen Kontakten abgefragt werden müssen), ist dort die Möglichkeit, einzelne Bits mit einem einzigen Befehl auf ihren momentanen Wert testen, sie setzen oder löschen zu können, extrem wichtig. So bewirkt z. B. der Befehl

SET 4, C

daß im Register C das Bit Nr. 4 (also das 5. bit von der niederwertigsten Stelle Ø aus gesehen) auf den Wert 1 gebracht wird während alle übrigen bits unverändert bleiben.

i) Kombinierte Adressierweisen

An dieser Stelle muß angemerkt werden, daß die Leistungsfähigkeit eines Prozessors wesentlich von der Möglichkeit aller genannten Adressierweisen mit bestimmt wird. Noch weiter gesteigert wird die Effizienz dann verständlicherweise, wenn Kombinationen mehrerer Adressierweisen möglich sind. Als Beispiel soll der folgende Z80–CPU–Befehl dienen:

BIT 2, (IY+0DH)

indem indizierte und Bit-Adressierweise miteinander verknüpft sind. Er bewirkt das Testen des Bit Nr. 2 in der Speicherstelle, deren Adresse durch den Inhalt des Zweit–Indexregisters IY plus einer Distanz von 13 (= hexadezimal 0D) gegeben ist.

j) symbolische Adressierung

Bei den bisher dargestellten Adressierungsmethoden werden die physikalischen Speicheradressen beim Aufbau eines Programms vom Programmierer entweder als effektive Speicheradressen (direkte Adressierung) oder mittels relativer, indizierter bzw. indirekter Adressierung festgelegt.

Diese Verfahren erfordern eine sehr exakte Arbeitsweise und genaueste Maschinenkenntnisse. Sie sind nicht immer zwingende Voraussetzung bei der Programmierung. Wesentlich ist vielmehr, daß in einem Programm neben den Instruktionen Datenfelder bestimmter Länge und Eigenschaften sowie Konstanten bestimmten Inhaltes eindeutig definiert werden. Es genügt dann, die eigentlichen physikalischen Speicheradressen für derartig definierte Größen durch symbolische Namen zu ersetzen. Für solche Namen werden zur besseren Übersicht und Merkfähigkeit mnemotechnische Kurzbezeichnungen (sog. symbolische Adressen) gewählt.

Beispiel: SUMME für das Ergebnis einer Addition mit zwei Operanden A und B

Da der Zentralprozessor selbst faktisch nur mit Binärziffern arbeitet, nicht aber Symbole interpretieren kann, sind zu dieser Adressierweise (wie auch für jede symbolische Programmierung, wie wir sie schon in Form der in diesem Abschnitt gezeigten Beispielbefehle kennengelernt haben) Hilfsprogramme nötig, die das DECODIEREN und ZUORDNEN dieser Symbole (symbolischer Op Codes und symbolischer Adressen) auf die Bit-Ebene besorgen.

Bei der Umwandlung des vom Programmierer in einer bestimmten Programmiersprache geschriebenen Primärprogramms (Quellenprogramm) in ein echtes Maschinenprogramm (Objektprogramm) werden vom Umwandlungsprogramm (Assembler oder Compiler) diese symbolischen Adressen in einer Tabelle registriert und ihnen dann unter Berücksichtigung der gegebenen Definition effektive Speicheradressen zugeordnet (häufig: relative Adressierung); prinzipiell kann man dies natürlich auch mit Hilfe der nötigen Tabellen "von Hand" machen. Mit dieser Problematik werden wir uns in Abschnitt 5 eingehend befassen.

2.5. Die Befehlsverarbeitung des Zentralprozessors

In den bisherigen Abschnitten haben wir erfahren, wie ein Zentralprozessor grundsätzlich aufgebaut ist, in welcher Form er Befehle und Daten empfängt und welche Möglichkeiten zur Adressierung der Speicherzellen von Befehls- und Daten-Speicher es gibt.

Es soll jetzt im Einzelnen gezeigt werden, wie die Zentralprozessor-Hardware die Befehle "ver"-arbeitet. Da einerseits die im Handel befindlichen Mikroprozessoren dies in unterschiedlicher Weise tun, andererseits hier konkretes Wissen vermittelt werden soll, muß dazu ein bestimmter Prozessor beispielhaft behandelt werden; dieses Beispiel ist auf andere Prozessortypen sinngemäß übertragbar. Ausgewählt wurde der modernste 8bit-Universal-Mikroprozessor, der Z80-CPU der Firma ZILOG, dessen Blockschaltung wir bereits in Bild 4 kennengelernt hatten.

Eine detaillierte Darstellung seiner Register zeigt Bild 5, insgesamt enthalten sie 208 dem Anwender zugängliche Bits, die ausgelesen oder geschrieben werden können.

Bild 6: Schematische Speicherdarstellung (bei Festwertspeichern gibt es natürlich statt der Lese/Schreib-Logik nur eine Auslese-Logik)

Hiervon sind zwei gleichaufgebaute Blöcke mit je 6 allgemeinen Registern, die jeweils wahlweise 8- oder 16-bit-Operationen erlauben.
Hinzu kommen zwei Akkumulatoren und Status ("Flag")-Register.
Die Operationen der CPU laufen in einem der beiden Register/Akku-Blöcke ab; der Zugriff zum zweiten Register/Akku-Block erfolgt durch Übergangs- ("Exchange"-) Anweisungen.
Diese alternative Arbeitsweise ermöglicht wechselweises Arbeiten im Haupt- und Hintergrundprogramm ohne Auslagern von Registerinhalten in den Arbeitsspeicher.
Diese Register sind das "Kurzzeitgedächtnis" der CPU, während größere Datenmengen (und natürlich auch das ganze Anwenderprogramm) in separaten Speicherbausteinen abgelegt sind; wir haben bereits in Abschnitt 2.4. gesehen, daß man sowohl Register (was sehr rasch geht) als auch Speicherstellen (was wesentlich mehr Zeit braucht) adressieren kann.
Da sich sämtliche Befehle in externen Speicherbausteinen befinden, sieht jeder Befehlsablauf (="Befehlszyklus"="Instruction Cycle") im Prinzip gleich aus, und zwar so, wie in Bild 7 dargestellt.

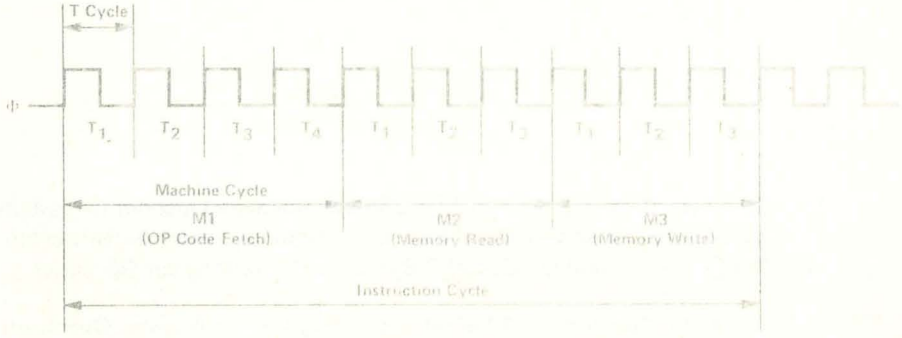


Bild 7: Darstellung eines Befehlszyklus

Man sieht daraus, daß folgende Hierarchie besteht:

- Befehlszyklus: Zeit zur Ausführung eines Befehls, besteht aus
- Maschinenzyklen: die Anzahl der Maschinenzyklen pro Befehl ist durch die Eigenschaften des einzelnen Befehles gegeben. Jeder Maschinenzyklus besteht seinerseits aus mehreren
- Taktzyklen: Die Dauer der Taktzyklen ist durch die Frequenz des Taktgenerators gegeben. Die Taktzyklen-Zeit bei der Z80 –CPU beträgt ~~1,6 usec~~ (≙ 2,5 MHz), bei der Z80A–CPU ~~1 usec~~ (≙ 4 MHz). 400 nsec. 250 nsec

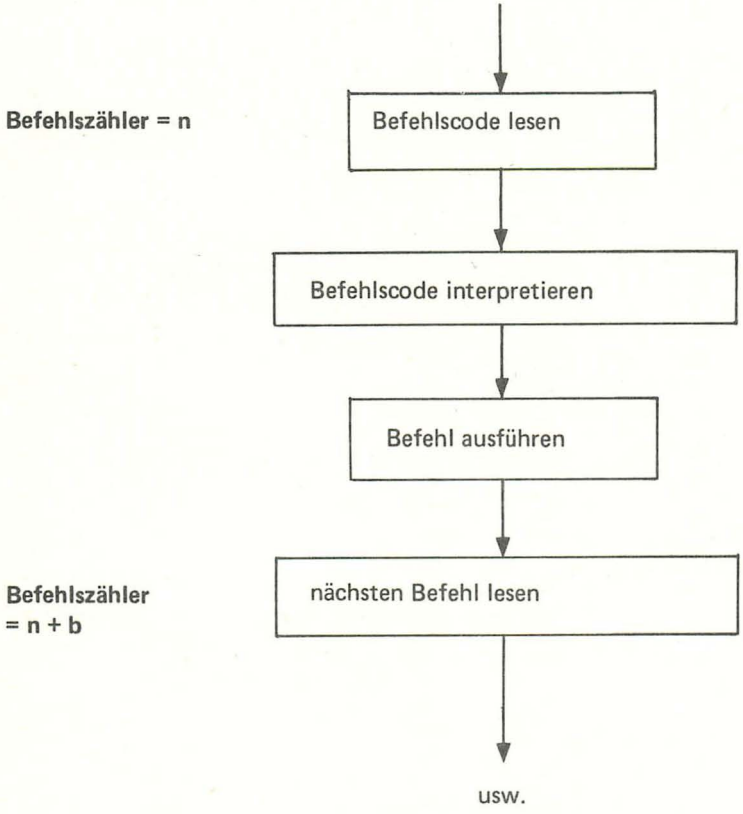
Die Länge eines Maschinenzyklus beträgt zwischen drei und sechs Taktzyklen; ist die Länge eines Maschinenzyklus (z.B. des M1, bei dem der Befehlscode aus dem Programmspeicher gelesen wird) für die maximal zulässige Arbeitsgeschwindigkeit der verwendeten Speicherbausteine zu kurz, kann sie durch Einfügen eines zusätzlichen Taktzyklus über den WAIT–Eingang der CPU verlängert werden.
Beim Maschinenzyklus M1 wird der Code des nächstfolgenden Befehls aus dem Programmspeicher in die CPU eingelesen und decodiert.
Da die einzelnen Befehle aus einer unterschiedlichen Anzahl von Bytes bestehen, wie wir in Abschnitt 2.4 gesehen haben (nämlich je nachdem, wie lang der Opcode ist – 1 oder 2 Byte*) und verschieden lange Adreßfelder haben (0 . . .2 Byte**), ergeben sich Gesamt-Befehlslängen von (1 . . .2) + (0 . . . 2) = max. 4 Byte. Da jedes einzelne Befehls-Byte aus dem Speicher in die CPU eingelesen werden muß, ergeben sich konsequenterweise auch unterschiedliche Längen des M1-Zyklus (4 . . . 6) Taktzyklen).
Zu beachten ist, daß beim M1 - Zyklus außer dem Einziehen des Opcodes auch dessen CPU-interne Decodierung erfolgt. Während dieser Zeit ist die CPU mit den Bussen nicht in Kontakt, so daß währenddessen andere Vorgänge wie DMA und Refresh abgewickelt werden können, worauf wir später noch zurückkommen werden.

* Da mit einem Byte = 8 bit nur 2^8 = 256 verschiedene Opcodes dargestellt werden können, brauchen Maschinen mit umfangreicheren Befehlssätzen wie Z80-CPU (über 600 Op-Codes) 2 Bytes zur Darstellung der Op-Codes.

** 0 Byte: Kein Operand im Adreßfeld (z.B. Register — Register-Operation, die bereits die Registeradressierung im Opcode beinhaltet).

1 Byte: 8bit-Operand im Adreßfeld
2 Byte: 16bit-Operand im Adreßfeld

Die auf M1 folgenden Maschinenzyklen M2 und M3 dienen zum Datentransfer zwischen dem Zentralprozessor einerseits und Speicher- oder Ein/Ausgabe-Bausteinen andererseits.
Wir erhalten also für die Befehlsausführung den in Bild 8 dargestellten schematischen Ablauf. Bitte machen Sie sich klar, daß dadurch zwar eine grundsätzlich sequentielle Verhaltensweise des Systems vorgegeben ist, aber einfach durch Verändern des Befehlszähler-Inhalts "Sprünge" möglich sind.



(dabei ist b die Länge des soeben ausgeführten Befehls in Bytes).

Bild 8: Schematischer Ablauf einer Befehlsausführung

Welche Aktivitäten der Zentralprozessor nach dem Einlesen eines Befehls durchführt (M2 und M3) hängt vom Opcode bzw. dessen Decodierung in der CPU ab:

- Worttransfer-Befehle
  Informationen werden zwischen den internen Registern der CPU oder zwischen den Registern der CPU und externen Speicherzellen übertragen. Bei all diesen Befehlen ist eine Sendeadresse und eine Zieladresse entsprechend den in 2.4. gegebenen Regeln spezifiziert; dabei wird der Inhalt der Sendeadresse grundsätzlich nicht geändert.

  Beispiel: LD C, B

  kopiert den Inhalt des Registers B in das Register C, ohne den Inhalt von B zu verändern.
  Hierzu gehören auch Direkt-Ladebefehle, die Konstanten in ein Register oder eine Speicherzelle laden, und Befehle, die zwei Registerinhalte austauschen.

- Blocktransfer-Befehle: Eine besondere Eigenschaft des Z80 -CPU–Mikroprozessors ist seine Fähigkeit, mit einem einzigen Befehl einen ganzen Datenblock beliebiger Länge zu übertragen.
  Da praktisch bei jeder Anwendung Datenpakete mehr oder minder großer Länge transferiert werden müssen, trägt dieser Befehlstyp besonders zur Einsparung von Speicherplatz (bei n-Byte Blocklänge braucht der Transferbefehl nicht n-mal sondern nur 1mal abgespeichert zu werden) und Befehlsausführungszeit bei (8,4 μ sec/Byte).
  Ebenso wie Block-Transfers ist das Aufsuchen eines Suchbegriffs in einem Datenblock beliebiger Länge mit einem einzigen Befehl möglich.

- Arithmetisch/logische Befehle
  arbeiten mit Datenwörtern, die sich in CPU-Registern oder den externen Speicherbausteinen befinden.
  Dabei wird (außer bei Inkrementier- und Dekrementier Befehlen) das Ergebnis der Operation jeweils in den Akkumulator abgespeichert und die Bedingungsbits (= "Flags") entsprechend dem Wert des Ergebnisses gesetzt.
  Ein Beispiel hierfür ist die Addition einer Speicherstelle auf den Akkumulator-Inhalt, bei der das Ergebnis wieder im Akku steht.
  Zu dieser Befehlsgruppe gehören neben den 8bit-Befehlen auch 16bit-Additions- und Subtraktionsbefehle.

- Bit-Manipulationsbefehle:
  Mit ihnen kann mit einem einzigen Befehl ein beliebiges Bit im Akkumulator, einem allgemeinen Register oder einer Zelle des Arbeitsspeichers gesetzt (Wert = 1), rückgesetzt (Wert = 0) oder auf seinen Inhalt getestet werden.
  Damit sind besonders gut Maschinensteuerungsprobleme zu lösen, aber auch Aufgaben in allgemeinen Programmen, in denen Statusinformationen abgefragt und beeinflußt werden müssen.

- Verzweigungs-Befehle:
  Hierzu gehören Sprungbefehle, Befehle zum Aufruf von Unterprogrammen und Befehle zum Rücksprung von Unterprogrammen ins Hauptprogramm.
- Ein/Ausgabe-Befehle
  wickeln den Datentransfer zwischen Speicherstellen bzw. CPU-Registern einerseits ab. Sie bringen die Statusbits (="Flags") in die entsprechenden Zustände, so daß beispielsweise eigene Operationen zur Paritätsprüfung überflüssig werden.
  Die Z80-CPU kann übrigens auch Block-Ein/Ausgaben einer Länge bis zu 256 Bytes mit einem einzigen Befehl ausführen.
- CPU-Steuerbefehle
  geben der CPU die gewünschten Betriebsarten vor.

2.5.3. Bedingungsbits (="Flags")

Die Z80-CPU verfügt über zwei Status- (Flag) Register, die über die Art des Ergebnisses der letzten Prozessoroperationen Auskunft geben. Diese Auskunft wird im Wesentlichen dazu benützt, bedingt Sprünge vorzunehmen, d.h. je nach Ergebnis der Prüfung eines dieser Bedingungsbits die eine oder aber eine andere Aktion durchzuführen. Diese Verzweigung geschieht durch sogenannte Sprungbefehle, die im System Z80 relative oder aber auch absolute Sprünge sein können.

Die Stellung der einzelnen Bedingungsbits innerhalb des Flag-Registers zeigt folgende Skizze:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

Hierbei bedeutet:

C: Überlaufsbit (Carry-Flag)

N: Additions / Subtraktionsbit (Add / Subtract-Flag)

P/V: Paritäts / Überlaufbit (Parity / Overflow-Flag)

H: Halb-Byte-Überlaufbit (Half-Carry-Flag)

Z: Null-bit (Zero-Flag)

S: Vorzeichen-bit (Sign-Flag)

X: nicht verwendet

Die beiden Z80-CPU-Bedingungsbit-Register enthalten also 6-bit Status-Informationen, die auf Grund des Ergebnisses der vorhergegangenen CPU—Aktivität auf 0 oder auf 1 gesetzt werden; die Bits in der Pos. 3 und 5 der Bedingungs-bit-Register sind unbenützt.

Vier der genannten Bit lassen sich für bedingte Sprünge, bedingte Unterprogrammaufrufe oder bedingte Rücksprungbefehle testen. Es sind dies die Flags C, P/V, Z und S. Die beiden Flags H und N werden intern für die Abwicklung von BCD — Operationen verwendet und lassen sich vom Programmierer nicht direkt verwenden. Auf die Bedeutung der einzelnen Bedingungsbits soll hier im Einzelnen eingegangen werden.

2.5.3.1. Das Überlaufbit (Carry-Flag C).
Das Setzen bzw. Rücksetzen der Carry-Flag wird je nach Art der durchgeführten Operation verschieden behandelt. So wird bei Additionsbefehlen, bei denen ein Übertrag entsteht, und Subtraktionsbefehlen, an denen kein negativer Übertrag entsteht, die Carry-Flag gesetzt.
Rückgesetzt wird die Carry-Flag bei Additionsbefehlen, die einen negativen Überlauf ergeben. Durch dieses Hilfsmittel lassen sich leicht Programme schreiben, die arithmetische Operationen mit Wortlängen größer ein Byte durchführen. Darüber hinaus wird das Carry-Flag benützt, um bei BCD - bzw. 4-bit-Operationen anzuzeigen, daß die Bedingungen für die Durchführung der Dezimalumwandlungen gegeben sind. Bei den Rotationsbefehlen RLA, RRA, RLS und RRS wird das Carry-bit als Zwischenspeicher für die Übertragung des niederwertigsten und höchstwertigsten Bits für irgendeines der CPU—Register— bzw. Speicherstellen benützt. Bei den Befehlen RLCA, RLC s und SLA s enthält das Carry-Flag den Wert des höchstwertigen Bits (bit 7) des durch den Befehl behandelten Registers bzw. der Speicherstelle, der aus diesem Register hinausgeschoben wurde.

Bei den Befehlen RRCA, RRC s, SRA s und SRL s enthält das Carry-Bit den zuletzt aus den niederwertisten Bit des betreffenden Registers bzw. der Speicherstelle hinausgeschobenen Bytes.

Bei den logischen Befehlen AND s, OR s und XOR s wird das Carry-Flag grundsätzlich zurückgesetzt.

Durch die eigens hierfür implementierten Befehle SCF (Set-Carry-Flag) und CCF (Complement-Carry-Flag) kann das Carry-Flag direkt gesetzt und komplementiert werden.

2.5.3.2. Das Additions- und Subtraktionsbedingungs-bit (N)
wird bei dem Befehl zur Dezimalumwandlung des Akkumulators (DAA) benützt, um zwischen Additions- und Subtraktionsbefehlen zu unterscheiden. Bei allen Additionsbefehlen wird N rückgesetzt bei Subtraktion auf 1 gesetzt.

2.5.3.3. Das Paritäts / Überlauf-Bedingungs-bit (P/V) ;
Auch dieses Bedingungsbit wird von verschiedenartigen Befehlen in verschiedenartiger Weise beeinflußt und benützt. Bei arithmetischen Operationen zeigt dieses Bedingungsbit einen Überlauf an, sobald das Ergebnis im Akkumulator größer als die maximal zulässige positive Zahl (+127) oder kleiner als die höchste zulässige negative Zahl (-128) ist.Welcher der beiden genannten Fälle gegeben ist, läßt sich durch Prüfen des Vorzeichenbits der beiden Operanden feststellen: Bei Addition von Operanden mit unterschiedlichem Vorzeichen kann natürlich grundsätzlich kein Überlauf entstehen. Sobald dagegen Operanden gleichen Vorzeichens addiert

werden und dabei ein umgekehrtes Vorzeichen entsteht, so wird das Überlauf-Flag gesetzt. Betrachten Sie hierfür bitte folgendes Beispiel:

+120 =	0111	1000	ADDEND
+105 =	0110	1001	AUGEND
+225	1110	0001	(–95) SUM.

In diesem Beispiel bringen die beiden miteinander addierten Zahlen ein Ergebnis, das größer als +127 ist, wodurch ein binäres Ergebnis erzeugt wird, das ein negatives Vorzeichen hat (-95), was natürlich nicht richtig ist. Aus diesem Grund wird das Überlaufbedingungsbit an dieser Stelle gesetzt. Im Gegensatz hierzu kann bei Subtraktionen ein negativer Überlauf nur dann vorkommen, wenn die beiden Operanden ein unterschiedliches Vorzeichen haben. Betrachten Sie hierzu folgendes Beispiel:

	+127	0111	1111	MINUEND
(–)	–64	1100	0000	SUBTRAHEND
	+191	1011	1111	DIFFERENZ

In diesem Beispiel wurde wiederum eine Vorzeichenänderung bewirkt, die ein unrichtiges Ergebnis zustande bringt.
Die Carry-Flag wird außerdem benützt bei logischen Operationen und Rotationsbefehlen, um die Parität des Ergebnisses zu überprüfen; dabei wird die Anzahl der einzelnen Einsen im angesprochenen Byte gezählt.

Ist deren Anzahl dagegen gerade, so wird das Paritäts-bit gesetzt (P=1). Bei Blocktransfer (LDI, LDIR, LDD, LDDR) und Blocksuchbefehl (CPI, CPIR, CPDR) gibt die P/V-Flag Auskunft über den Stand des Byte-Zählers (Registerpaar BC). Beim Dekremenieren wird dieses Flag auf 0 gesetzt, wenn der Byte-Zähler das Ergebnis 0 bringt; in allen übrigen Fällen ist der Wert des Bedingungs-bit 1. Bei Ausführung der Befehle LD A, I und LD A,R wird der Inhalt des Interrupt-Enable-Flip-Flops zu Testzwecken in das P/V-Flag übertragen.

Bei Eingabe eines Bytes von einem Eingabegerät über den Befehl IN r, (C) gibt das Paritäts-Flag Auskunft über die Parität der eingezogenen Daten.

2.5.4. Das Halb-Byte-Übertragsbit (H)

wird je nach dem Übertragsergebnis zwischen den Bits 3 und 4 einer beliebigen 8bit arithmetischen Operation gesetzt oder rückgesetzt. Verwendet wird diese Flag bei dem Befehl zur BCD-Wandlung (DAA) im Akkumulator um das Ergebnis einer gepackten BCD Addition bzw. Subtraktion zu korrigieren.

Das H-Bedingungsbit wird entsprechend der folgenden Tabelle gesetzt bzw. rückgesetzt:

H	Addition	Subtraktion
1	kein Übertrag von bit 3 zu bit 4	kein negativer Übertrag von bit 4
0	kein Übertrag von bit 3 nach bit 4	ein negativer Übertrag von bit 4

2.5.3.5. Das Null-Bedingungsbit (Z)

Bei einer Reihe von Befehlen wird das Nullbit (Zero-Flag Z) ensprechend dem Ergebnis der jeweiligen Operation gesetzt oder rückgesetzt.

Bei 8bit arithmetischen und logischen Operationen wird das Null-Bedingungsbit auf 1 gesetzt, wenn das Ergebnisbyte des Akkumulators gleich 0 ist. Ist das Ergebnis nicht = 0, so wird das Zero-Flag auf den Wert logisch 0 zurückgesetzt.
Bei Vergleichs- und Suchbefehlen wird das Null-Bedingungsbit auf 1 gesetzt, sobald der Vergleich zwischen dem im Akkumulator befindlichen Wert und der Speicherstelle, die durch den Inhalt des Registerpaares HL adressiert wird, positiv ausgefallen ist.
Beim Testen eines Bits in einem Register oder einer Speicherstelle wird dem Null-Bedingungs-bit der komplementäre Wert der getesteten Bitstelle übertragen.
Bei der Übertragung eines Bytes zwischen einer Speicherstelle und einer Ein/Ausgabe-Schnittstelle (INI, IND, OUTI und OUTD) wird das Zero-Flag gesetzt, sobald der Wert des Zählregisters (Register B) zu 0 wird. Darüber hinaus wird das Zero-Flag auf Eins gesetzt, wenn über den Befehl IN r, (C) eingezogenes Byte den Wert "0" hat.

2.5.3.6. Das Vorzeichen-Bedingungsbit (Sign-Flag)

In das Vorzeichen-Bedingungs-bit (S) wird bei bestimmten Befehlen das höchstwertigste Bit des Akkumulatorinhalts kopiert. Bei der Ausführung von arithmetischen Befehlen mit vorzeichenbehaftenden Zahlen wird die binäre Zweier-Komplement-Darstellung verwendet. Eine positive Zahl ist durch eine "0" im höchstwertigen bit eines Bytes (Bit 7) identifiziert, eine negative Zahl durch eine "1" in dieser Bit-Stelle.

2.6. Schlaglichter auf Mikrocomputer-Funktionen

2.6.1. Ob 'Keller' oder "Stapel" - das Stack ist wichtig

Für Aufgaben , die man immer wieder durchführen muß, überlegt man sich am besten eine Routine – das gilt für das tägliche Leben genauso wie für Computer-

Software; d.h. das sich wiederholende Problem soll in immer gleicher Weise, ohne irgendwelchen Aufwand im Einzelfall, erledigt werden.
Da solche Routinen (oder "Unterprogramme") an beliebigen Stellen verwendbar sein müssen, müssen bei der Verwendung (="Aufruf") definierte und reproduzierbare Verhältnisse geschaffen werden, damit nach der Routine der Ablaufrahmen (="Hauptprogramm") korrekt weiterverfolgt werden kann.
Man tut dies in Mikrocomputer am zweckmäßigsten, indem man sämtliche Informationen, die nach Erarbeitung des Programms zur Weiterverfolgung des Hauptprogramms nötig sind, in einen reservierten Speicherbereich auslagert oder "rettet" und sie bei Abschluß des Unterprogramms automatisch **dahin** zurückbringt, wo sie hingehören.

Dieser reservierte Speicherbereich heißt "Stapelspeicher" oder "Kellerspeicher' oder "Stack".
Dorthin transferiert man insbesondere

- den momentanen Stand des Befehlszählers (das ist, um genau zu sein, immer die Adresse **des** Befehls, der als **nächstes** ausgeführt wird). Dies geschieht durch den Befehl zum Aufruf eines Unterprogramms ("CALL") automatisch.
- CPU-Register, die vom Unterprogramm in irgendeiner Form benutzt werden; das ist praktisch immer der Akkumulator und das Zustands ("Flag")-Register, meist zusätzlich noch weitere Register.

Dieses "Retten" bzw. Auslagern dieser Registerinhalte wird durch den Befehl PUSH veranlaßt und geschieht registerpaarweise, d.h. jeder PUSH-Befehl bringt zwei 8bit-Register in den Stapelspeicher (z.B. Registerpaar AF, HL, u.s.w.). den umgekehrten Vorgang bewirkt der POP-Befehl. Von ihm werden jeweils zweimal 8bit aus dem Stapelspeicher in das entsprechende Registerpaar der CPU geladen.
Man wird also bei der Programmerstellung grundsätzlich folgendes Schema verfolgen:

```
----------      ; HAUPTPROGRAMM

CALL SUBROU     ;AUFRUF DES UNTERPROGRAMMS,
                 BEFEHLSZAEHLER WIRD AUTOMATISCH
                 GERETTET
PUSH AF         ; AKKU UND FLAGS RETTEN
PUSH BC         ; REGISTER B UND REGISTER C RETTEN
                ; UNTERPROGRAMM AUSFUEHREN
POP BC          ; REGISTER B UND REGISTER C MIT ALTEN
                ; WERTEN LADEN
POP AF          ; AKKU UND FLAGREGISTER WIEDER MIT
                 ALTEM WERT LADEN
RET             ; RÜCKSPRUNG IN HAUPTPROGRAMM D.H.
                 ALTEN BEFEHLSZAEHLERSTAND WIEDER
                 LADEN

---------       ; HAUPTPROGRAMM WEITER AUSFUEHREN
```

Das vorherige Beispiel zeigt die unbedingt erforderlichen Befehle für den Aufruf eines Unterprogramms, das den Akku und die Register B und C benutzt.

Programmier-Merkregel 1:
Wenn man mit dem Programmspeicherplatz nicht knapp ist, sollte man es sich angewöhnen, bei Unterprogrammaufrufen grundsätzlich **sämtliche** Register über den Stapelspeicher zu "sichern".
Der Aufwand ist nicht groß; pro Registerpaar kostet das 2 Bytes Programmspeicher für allgemeine Register (PUSH=1 Byte + POP = 1 Byte) bzw. 4 Bytes für die Index-Register, und Probleme bei Programmänderungen (denken Sie daran, daß Sie vielleicht einen Befehl in ein Unterprogramm schreiben, das auf ein bisher unbenutztes Register zugreift, ohne daran zu denken, daß auch ein PUSH/POP-Befehlspaar eingefügt werden muß!) werden zuverlässig vermieden.

Hardwaremäßig läuft übrigens der beschriebene Vorgang nach dem "Last In-First Out"-Prinzip ab; das bedeutet, daß die zu rettenden Informationen sukzessive in den Stapelspeicherbereich hineingeschoben werden, dessen Umfang also proportional zur Anzahl der gleichzeitig im Stapelspeicher abgelegten Informationen wächst. Dieses Wachsen geschieht in Richtung niederwertigerer Adressen.
Aus diesem Grund legt man den Anfang des Stapelbereichs immer auf sehr hohe Adressen im Schreiblesespeicher.

Programmier-Merkregel 2:

Da die Länge des Stapelspeichers nicht von vornherein zu limitieren ist, ist es sinnvoll, seinen Beginn so zu legen, daß unbeabsichtigtes Wachstum des Speichers Variablen-Werte des Programms nicht zerstört.
Dies geschieht, indem man die Variablen-Adressen ans oberste Ende des Schreiblese-Speichers legt und darunter den Anfang des Stapelspeichers festlegt (Bild 9).

Da der übrige Adreßbereich meist unbenützt oder aber Festwertspeicher ist, können Stack-Aktivitäten dann keine Daten zerstören und evtl. durch Stack-Überlauf verursachte Fehler sind viel leichter erkennbar.

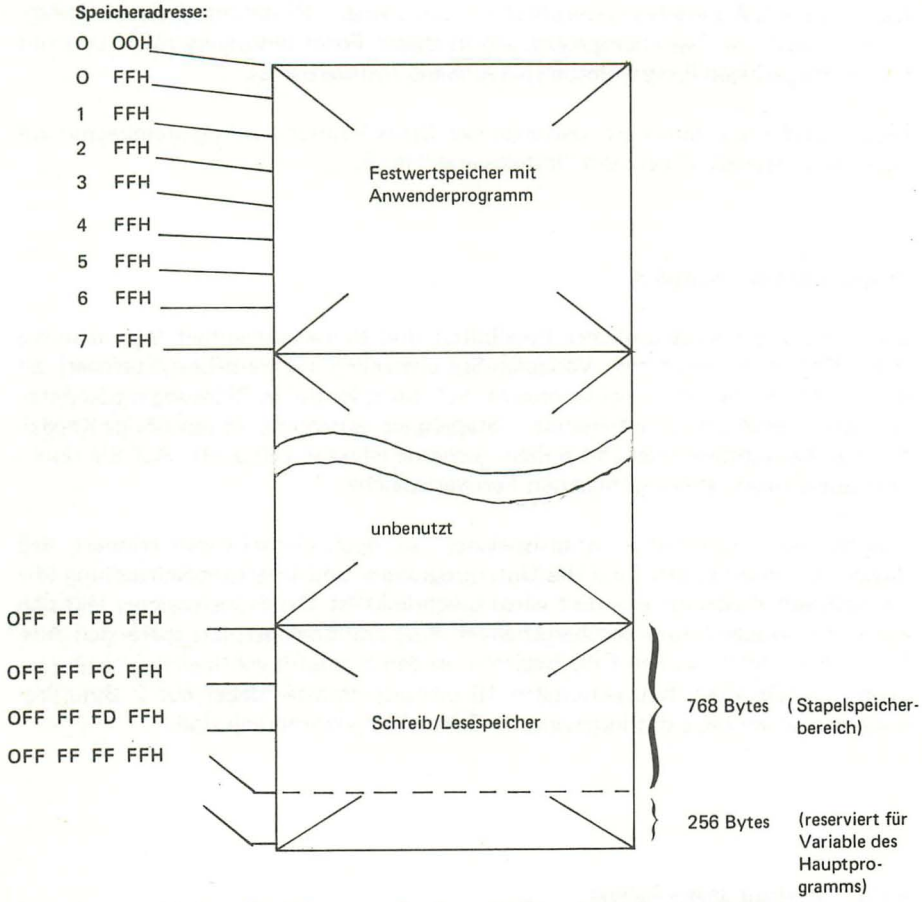


Bild 9: Beispiel einer Speicherorganisation mit Stapelspeicherbereich

Es wurde bisher stillschweigend angenommen, daß der Zentralprozessor **weiß,** wie er den Stapelspeicherbereich zum Einschreiben ("PUSH") und zum Auslesen ("POP) behandeln muß. Dieses "Wissen" liegt in einem seiner Spezialregister, dem "Stapelzeiger" (="Kellerspeicherzeiger" bzw. "Stack-Pointer"). Dieses 16-Bit-Registergister wird vom Steuerwerk der CPU immer so versorgt, daß es die Speicher-Adresse enthält, die über eine Stapelspeicher-Operation als nächste angesprochen wird. Ein PUSH-Befehl bewirkt also, daß der Inhalt eines Registerpaars so in einen vorher festgelegten Teil des Schreib/Lesespeichers aus gelagert wird, daß er durch einen Pop-Befehl ohne irgendwelche zusätzliche Manipulationen in das gleiche Registerpaar wieder eingelesen wird. Dies ist, wie wir schon gesehen haben, zum "Retten" der Registerinhalte bei der Interrupt- oder Unterprogrammbehandlung wichtig, aber ebenso nützlich, wenn während des "normalen" Programmablaufs

Daten im RAM zwischengespeichert werden müssen. Hintergrund dieses Verfahrens ist, daß die Zwischenspeicherung in dieser Form besonders rasch und mit besonders geringen Programmspeicheraufwand realisierbar ist.

Dieses Verfahren impliziert, daß man den Stack-Pointer zu Programmbeginn auf einen Anfangswert setzen, ihn "initialisieren" muß.

Programmier-Merkregel 3:

Zur Erzielung höchstmöglicher Flexibilität und Betriebssicherheit legt man die vom Programm benützten Variablen-Speicherzellen (Schreib/Lese/Speicher) an das oberste Ende des Speicherbereichs; auf ihn folgt der in Richtung niederwertigere Speicheradressen "wachsende" Stapelspeicherbereich, dessen nötige Kapazität man genauestens oder mit hohem Sicherheitsfaktor kalkuliert. Auf die untersten Speicheradressen legt man den Festwertspeicher.

Abschließend zum Thma "Stapelspeicher" sei noch einmal daran erinnert, daß dessen Anwendung nicht auf die Unterprogramm- und Interrupt-Behandlung (die im nächsten Abschnitt erläutert wird) beschränkt ist. Der Stapelspeicher läßt sich auch hervorragend zum vorübergehenden Programmspeicherplatz-sparenden Auslagern von Daten aus den CPU-Registern in den Schreib/Lese-Speicher "mißbrauchen", da für einen bidirektionalen 16 bit-Datentransfer dabei nur 2 Byte Programmspeicherplätze mit insgesamt 21 Taktzyklen erforderlich sind.

2.6.2. Interrupt gegen Polling

Darunter versteht man die Möglichkeit, ein laufendes Programm (Hauptprogramm oder "Hintergrund" = "Background"-Programm) zu beliebigem Zeitpunkt von einer Interrupt-Anforderung "Interrupt-Request" unterbrechen zu lassen und ein anderes Programm, die "Interrupt-Service-Subroutine" einzuschieben. Ist dieses eingeschobene Programm abgearbeitet, setzt der Computer die Bearbeitung des Background-Programms fort.

Voraussetzung für diese Arbeitsweise ist natürlich, daß unmittelbar nach der Unterbrechung sämtliche momentanen Daten und Adreßinformationen des Haupt-Programms "gerettet" werden und nachher wieder bereitstehen. Dieses "Retten" geschieht ganz genauso, durch den Kellerspeicher, wie wir es in Punkt 2.6.1 bei der Unterprogrammbehandlung gesehen haben.
In einem Beispiel (Bild 10) soll der Interrupt-Vorgang klargemacht werden; es zeigt schematisch den Datenfluß in einem mikrocomputer-gesteuerten Fertigungsregelungssystem.

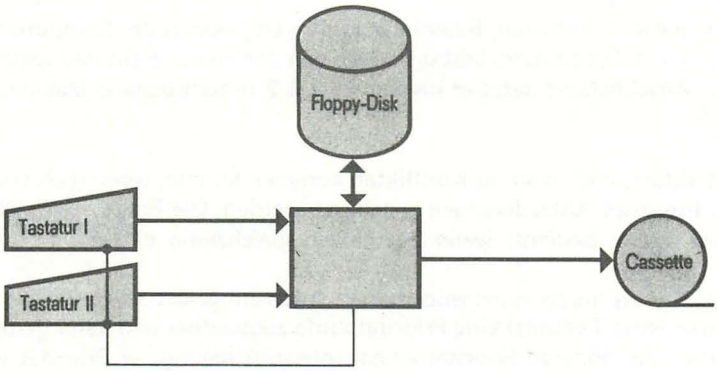


Bild 10: Beispiel eines Datenflusses in einem mikrocomputer-gesteuerten Fertigungssystem

Wir wollen annehmen, daß das Hintergrund-Programm aus sehr vielen, auf Floppy-Disk gespeicherten Daten zunächst einen Maximalwert suchen und dann noch andere arithemtische Auswertungen durchführen soll. Das Mikrocomputersystem muß aber währenddessen dauernd bereit sein, neue Daten, die irgendwann über Tastaturen eingegeben werden (z.B. Ergebnisse einer Fertigungsfortschrittskontrolle), zu empfangen und auf einem Magnetband-Cassettengerät abzuspeichern.

Gelöst wird das Problem durch Vorsehen eines Interrupts.

Im Bild 11 ist gezeigt, was passiert, wenn eine Dateneingabe von Tastatur zufällig erfolgt, während Befehl 4 - das Zwischenspeichern der zuletzt gefundenen größten Zahl - ausgeführt wird.

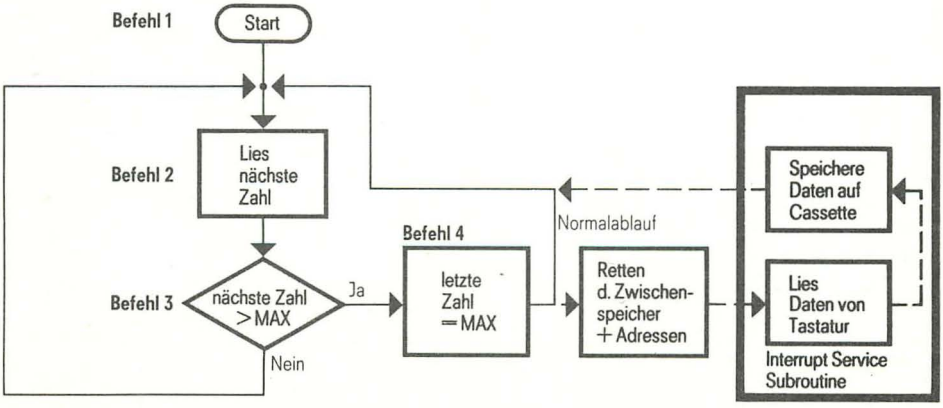


Bild 11: Schematischer Programmablauf bei Interrupt

Sobald Befehl 4 abgearbeitet ist, werden die wesentlichen Momentandaten des Hintergrund-Programms "gerettet" (= in Stack abgespeichert).

Statt nun, wie im Normalfall, Befehl 2 auszuführen, beginnt der Computer mit der Interrupt Service Subroutine, liest die Daten von der Tastatur ein und speichert sie auf Band. Anschließend setzt er die vor Befehl 2 unterbrochene Maximumsuche fort.

Man sieht sofort, daß es so zu Konflikten kommen könnte, wenn mehrere Terminals bzw. Interrupt-Anforderungen zugelassen werden. Die Frage ist: Welches Terminal wird zuerst bedient, wenn bei beiden gleichzeitig eine Eingabe erfolgt?

Das Problem wird durch einen priorisierten Interrupt gelöst; dabei wird jedem Interrupt (hier jeder Tastatur) eine Prioritätsstufe zugeordnet und dafür gesorgt; daß ein Interrupt der höheren Priorität einem Interrupt niedrigerer Priorität vorgezogen wird bzw. daß ein höherer Interrupt seinerseits einen niederen unterbricht und nicht umgekehrt; die niederpriorisierte Tastatur muß also einige Mikrosekunden warten, bis die höherpriorisierte Tastatur bedient ist. Erst dann werden ihre Daten vom Computer eingelesen. Die Priorisierung erfolgt durch Hardware-Maßnahmen, die wir in späteren Abschnitten noch eingehend behandeln werden.

Sobald ein Unterbrechen einer Interrupt-Service-Routine, die zu einer weniger hoch priorisierten Peripherie gehört, durch eine höher priorisierte Interrupt-Anforderung zugelassen ist, entsteht eine Interrupt-Schachtelung (Bild 12); bei jeder neuen Vorschachtelungsebene müssen die Inhalte der vom momentan laufenden Programm benützten Register in den Stapelspeicher gerettet werden.

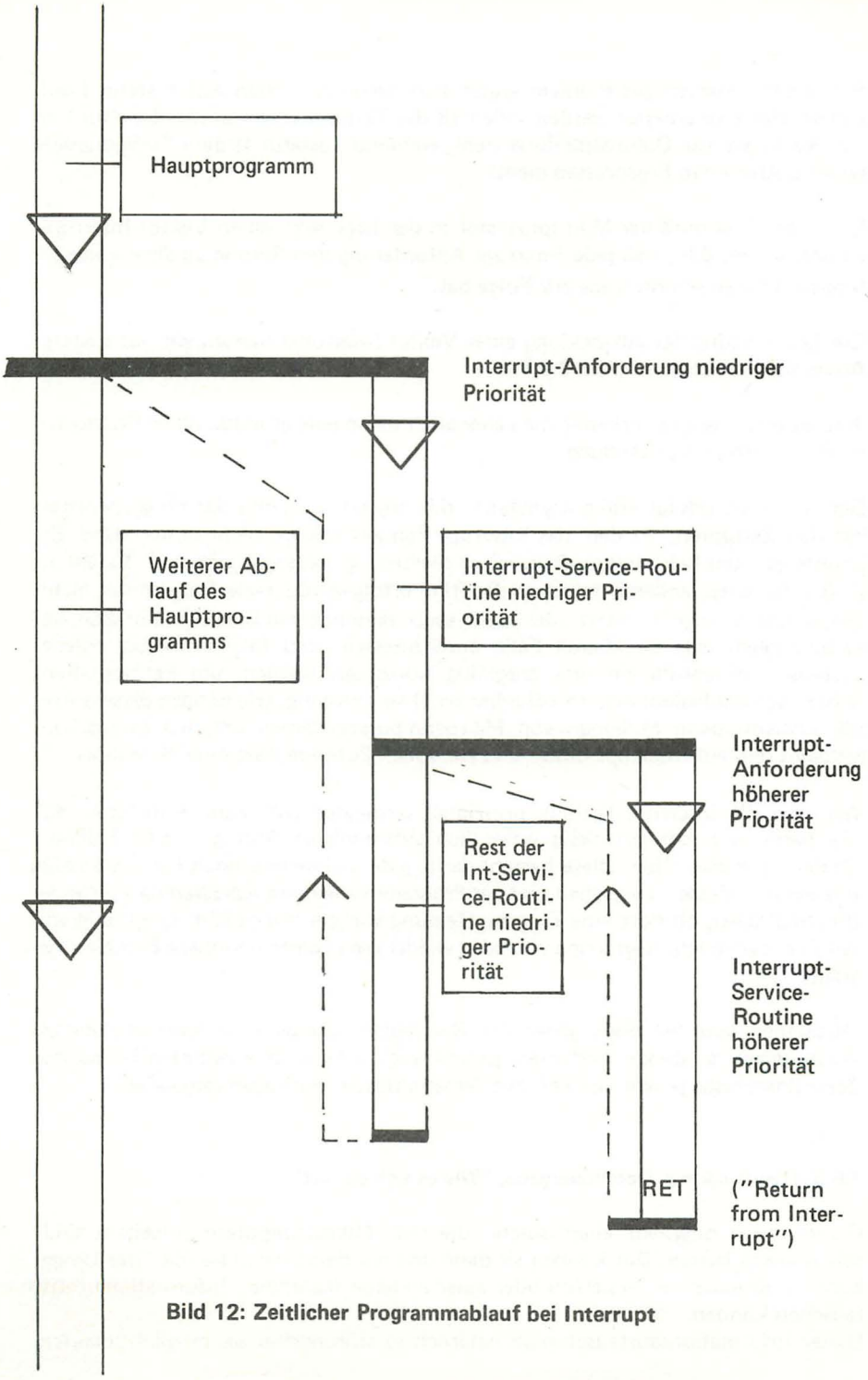


Bild 12: Zeitlicher Programmablauf bei Interrupt

Ein etwas andersartiges Problem ergibt sich, wenn die Daten von Tastatur I auf andere Weise verarbeitet werden sollen als die Tastatur II, wenn z.B. Tastatur I in der Werkstatt zur Datensammlung steht, während Tastatur II dem Fertigungsleiter zum Abruf von Ergebnissen dient.

In diesem Fall muß der Mikroprozessor in der Lage sein, einen Vektor-Interrupt zu bearbeiten, d.h., daß jede Interrupt-Anforderung den Sprung zu einer anderen Interrupt-Service-Subroutine zur Folge hat.

Die Besprechung der Abwicklung eines Vektor-Interrupts müssen wir auf spätere Abschnitte verschieben (Pkt. 2.9.1.).

Aus unserem Beispiel erkennt man aber auch schon eine grundsätzliche Problematik der Interrupt-Verwendung:

Der Interrupt erfolgt völlig asynchron, d.h. irgendwann, und der Programmierer hat den Zeitpunkt, zu dem das Interrupt-Request erfolgt, nicht in der Hand. Er könnte ja - um bei unserem Beispiel zu bleiben - genausogut während Befehl 1, 2, 3 oder einer anderen Interrupt-Routine erfolgen. Da reale Programme nicht vier, sondern einige hundert oder meist sogar tausende von Befehlen umfaßen, ist es unmöglich, alle denkbaren Fälle durchzutesten, und folglich müssen solche Systeme von vornherein sehr sorgfältig konzipiert werden, um Fehlverhalten sicher auszuschließen; dies ist offenbar nicht so schwierig, wie es nach diesem Abriß aussieht, denn Millionen von Mikrocomputersystemen arbeiten inzwischen weltweit rein auf Interrupt-Basis - und zur vollen Zufriedenheit ihrer Anwender.

Wer aber die Interrupt-Technik prinzipiell vermeiden will, kann stattdessen auf die hard- und softwaremäßig wesentlich aufwendigere Abfrage - oder Polling - Technik zurückgreifen. Diese besteht darin, jede periphere Einheit hardwaremäßig mit einer Adresse zu versehen und per Programm alle diese Adressen nacheinander durchzufragen, ob dort eine Eingabe-Meldung vorliegt oder nicht. Liegt eine vor, wird sie behandelt, liegt keine vor, wird wieder die nächste periphere Einheit abgefragt.

Abgesehen von den oben genannten Nachteilen und seiner Schwerfälligkeit bei Änderungen ist dieses Verfahren jedocn auch sehr rechnerzeitintensiv, da der Zentralprozessor ja sehr viel Zeit mit "ergebnislosen" Abfragen vergeudet.

2.6.3. Handshaking: Datenübergabe, "Wie es sich gehört".

Geräte- auch oder vor allem solche, die mit Mikrocomputern aufgebaut sind- sollen etwas **leisten**. Das können sie dann und nur dann, wenn sie mit ihrer Umgebung - z.B. einer Bedienperson oder einer anderen Maschine - Informationen austauschen können.
Dieser Informationsaustausch muß natürlich so störungsfrei wie möglich ablaufen.

Ein Verfahren, Informationen von einem "Sender" z.B. einem Mikrocomputer an einen "Empfänger " (z.B. einen Drucker) zu übertragen, ist das Quittungs- oder Handshaking-Verfahren. Es beruht im wesentlichen darauf, daß der Sender seine Informationen nicht "auf gut Glück" ausgibt, sondern erst dann, wenn gewisse Bedingungen auf der Empfängerseite erfüllt sind.
Die grundlegendste Bedingung hierfür ist die, daß der Sender erst dann senden darf, wenn der Empfänger empfangsbereit ist, also die zuvor gesendete Information "verdaut" hat; er signalisiert dieses dem Sender durch eine hierfür reservierte Leitung. Zuvor muß natürlich der Sender dem Empfänger mitgeteilt haben, daer ihm eine Nachricht übermitteln will.

Aus der Tatsache, daß dieser Informationsverkehr über die Bereitschaft zum Datentransfer nicht einseitig ist, sondern daß beide Kommunikationspartner sich über den Ablauf "einigen", ist die englische Bezeichnung "Handshaking" entstanden.

Der Vorgang kann nach verschiedenen Konventionen erfolgen; die im Einzelfall gültigen "Anstandsregeln" beim "Handshaking" sind Bestandteil der SchnittstellenDefinitionen, von denen einige weltweit standardisiert sind (z.B. RS 232, V 38 usw.). Als Beispiel soll hier die Schnittstellendefinition zwischen einem Z80-PIO-(Parallel-Ein/Ausgabe-Ports) Baustein und angeschalteter Peripherie dienen, die den meisten Standardschnittstellenanforderungen ohne zusätzlichen Hard- oder Software-Aufwand gerecht wird.

Man sieht, daß sich dort im Fall einer Ausgabe (die hier als Beispiel gezeigt wird) der Computerbaustein Z80-PIO zunächst mit einem READY-Signal meldet, das der angeschalteten Peripherie anzeigt, daß in der Z80-PIO Daten für sie bereitliegen. Nach ihrer Reaktionszeit quittiert die Peripherie dies durch Übermittlung eines Signals an die STROBE-Leitung der Z80-PIO und übernimmt die Daten.

Die Peripherie ist nun immer so aufgebaut, daß sie ihr Quittungssignal solange anstehen läßt, bis sie die Daten zuverlässig verarbeitet hat. Erst in diesem Moment nimmt sie ihre STROBE-Meldung wieder zurück, was wiederum der Z80-PIO das "o.K." für weitere Aktivitäten gibt.

Auf diese Weise ist sichergestellt, daß beim Informationsaustausch Sende- und Empfangs-Intentionen beider "Partner" nicht kollidieren.

2.6.4. Ein/Ausgabeschnittstellen

Wir haben bisher mehr oder minder stillschweigend angenommen, daß die Informationen, die aus dem Computer herauskommen oder ihm zugeführt werden, ähnlich wie bei den Bussen PARALLEL gehandhabt werden, d.h., daß z.B. beim Anschluß eines 8fach-Schaltersatzes tatsächlich 8 Leitungen in den Computer führen bzw. daß ein 8bit-ASCII-Zeichen, das auf einem Druckwerk gedruckt werden soll, diesem über 8 parallele Leitungen zugeführt wird.

Solche Aufgaben werden auch meist in der beschriebenen Form gelöst, und zwar mit Hilfe von sogenannten Parallel-Ein/Ausgabe-Bausteinen wie z.B. der Z80-PIO.

Diese Bausteine sind grundsätzlich an den Prozessor der gleichen Familie angepaßt und beinhalten folgende Funktionen:

- Trennung der Peripherie von den Systembussen. Das ist nötig, da die Peripherie (angeschlossene Schalter, AD-Wandler, Tastaturen, Anzeigen, Leistungsschalter) nicht mit der Aktivität der CPU synchronisiert sind, sondern ihre Signale IRGENDWANN bringen (Eingabe) respektive die Signale vom Prozessor AUCH DANN NOCH BRAUCHEN, wenn die CPU bereits mit anderen Informationen auf den Bussen arbeitet.
- Zwischenspeicherung "Latching" von Eingabe- und Ausgabe-Signalen aus denselben Gründen.
- Regelung des Datenverkehrs im Falle der bidirektionalen Datenübertragung (z.B. für Anschluß von Floppy Disks).
- Erzeugung und Bearbeitung von Quittungs (= "Handshaking")-Signalen (s. letzter Abschnitt) zur definierten und sicheren Datenübertragung.
- Erzeugung von Interrupt-Anforderungen an die CPU, Handhabung des Interrupt-Vektors und Priorisierung der Interrupt-Anforderungen der Peripherie.
- Sonderaufgaben, wie Einzelbitbearbeitung im Ein-Ausgabe-Baustein, Maskierung einzelnen Leitungen und Festlegung deren Aktivierungspegel (Low- oder High).

Solche Ein/Ausgabe-Bausteine umfaßen meist mehrere Kanäle (= "Ports").
Beim Anschluß von peripherer Hardware an solche Parallel-Ein/Ausgabe-Ports spricht man von "Parallel-Schnittstellen".
In vielen Fällen ist eine parallele Übertragung von Informationen jedoch unzweckmäßig, und zwar

- bei Anschluß der Peripherie über längere Leitungen aus Kostengründen (mehradrige Leitungen sind teuer und schlecht zu handhaben) und aus Störsicherheitsgründen (Verwendung von 1 verdrillten, evtl. abgeschirmten Leitpaar ist vorteilhafter).
- bei Anschluß von Geräten, bei denen serielle Datenübertragung die Norm oder sonst wie erforderlich ist (Fernschreiber, Telefon-Modem, Magnetkassettengerät, Floppy Disk usw.).

In diesen Fällen werden Serienschnittstellen eingesetzt. Die Mikrocomputer-Bausteine, die für jedes System verfügbar sind, werden busseitig genauso wie Parallel-Ein/Ausgabe-Bausteine behandelt, d.h. sie haben parallele Datenleitungen, die mit dem Datenbus direkt verbunden sind..

Die Parallel-Information des Adreßbus wandeln sie in eine Folge von Einzelbits um, die auf 1 Leitung nacheinander aus- bzw. eingegeben werden.

Über diese Umwandlungen hinaus haben sie die gleichen Aufgaben wie die oben beschriebenen Parallel-Ein/Ausgabe-Bausteine.

Da die serielle Datenübertragung häufig nach gewissen Regeln erfolgt, ist zusätzlich noch folgendes zu implementieren:

- Möglichkeit von synchroner (=taktgesteuerter) und asynchroner Arbeitsweise
- Variable Vergabe der Wortlänge, die jeweils seriell übertragen wird
- Erkennung von "Stop-Bits"
- Paritätssignal-Erzeugung und Prüfung zur Erkennung von Übertragungsfehlern
- Implementierung von Normprozeduren der seriellen Datenübertragung.

Aus dem vorher Gesagten und aus dieser Aufzählung wird klar, daß Serien-Ein/Ausgabe-Bausteine wesentlich komplexer und daher teuerer sind als Parallel-Ein/Ausgabe-Bausteine.

2.6.5. Zähler und Zeitgeber für Echtzeitereignisse

In allen Mikrocomputeranwendungen, die sich nicht auf reine algebraische Auswertungen beschränken, kommen zwei wesentliche Funktionselemente vor: Der

* Ereigniszähler, der auf 1 Leitung eintreffende Impulse zählt und bei einer per Programm festgelegten Anzahl dieser "Ereignisse" (=Impulse) eine Rechneraktivität (meist über Interrupt) veranlaßt.

  Beispiel:
  In einer Schrittmotorsteuerung soll eine vom Anwender-Programm ermittelte Schrittzahl n durchgeführt werden, bevor die Drehzahl-Abregelung erfolgt. Der Zähler gibt beim Erreichen dieses Zählerstandes eine Interrupt-Anforderung aus, die die "normale" Motoransteuerung des Hauptprogramms unterbricht und die Drehzahl-Abregelung veranlaßt.

* Zeitgeber, die eine vom Systemtakt (Quarz!) abgeleitete zeitlang ein Signal erzeugen, wobei diese Zeitdauer vom Anwenderprogramm kontrolliert wird.

  Beispiel:
  Pulslängenmodulation, bei der auf 1 Leitung ein Signal zu erzeugen ist, dessen Dauer einer vorgegebenen oder vom Anwenderprogramm errechneten Zahl entspricht.
  Im Prinzip lassen sich selbstverständlich beide eben beschriebenen Funktionen per Programm verifizieren:

a) der Zähler durch das Programm

```
ANFANG:   LD B, SCHLEI      ; VORGABE DER ANZAHL
                            ; DER ZU ZAEHLENDEN
          .                 ; EREIGNISSE
          .
          .                 ; HAUPTPROGRAMM

ZAEHL: PUSH AF              ; ANFANG DER ZUGEHOERIGEN
                            ; INTERRUPT-BEDIENROUTINE
       DJNZ ENDZAE          ; ZAEHLER  DEKREMENTIEREN
                            ; UND
                            ; AUF ROUTINENENDE
                            ; SPRINGEN FALLS DIE IN
                            ; REG B VORGEGEBENE
                            ; ANZAHL EREIGNISSE NOCH
                            ; NICHT STATTGEFUNDEN
                            ; HAT
          .                 ; NOETIGE AKTIVITAET
          .                 ; SOBALD VORGEGEBENER
          .                 ; ZAEHLERSTAND ERREICHT IST

       LD B, SCHLEI         ; VORGEGEBENE EREIGNIS-
                            ; ANZAHL IST EINGETROFFEN,
                            ; NEUE ZAEHLER INITIALISIEREN

       POP AF
ENDZAE: RETI                ; RUECKSPRUNG INS BACKGROUND - PROGRAM
```

wobei vorausgesetzt ist, daß ein Z80-PIO die Zählimpulse empfängt und bei jedem Zählerimpuls ein Anspringen der Interrupt-Routine ZAEHL veranlaßt, und

b) der Zeitgeber durch das Programm

```
ZEITSCH: PUSH AF            ; RETTEN ALLER REGISTER, DIE
         PUSH BC            ; VOM UNTERPROGRAMM
         PUSH DE            ; ZEITSC BENUETZT WERDEN
         XOR A              ; AKKU LOESCHEN
         SET ZBIT, A        ; BITPOS ZBIT AUF 1 SETZEN
ANF:     OUT (PORTD), A     ; SETZEN DES ZEITSIGNALS IN
                            ; PORTD
         LD BC, ZEIT        ; ZWEI 8-BIT-REGISTER MIT
                            ; ANZAHL ZAEHLSCHRITTE INI-
                            ; TIALISIEREN, DIE ZUM ERREI-
                            ; CHEN DER GEWUENSCHTEN ZEIT
                            ; ERFORDERLICH SIND
```

```
ZT1:    DEC B           ;B-REG DEKREMENTIEREN, BIS
        JR NZ, ZT1      ; ES NULL IST; DANN NAECHSTER
                        ; BEFEHL
        DEC C           ;C-REG DEKREMENTIEREN
        JR NZ, ZT1      ;SOLANGE B NICHT GLEICH
                        ; NULL IST, ERFOLGT WIEDERHO-
                        ; LUNG DER SCHLEIFEN B UND C
FERTIG:                 ;DIESER PUNKT WIRD ERST
                        ;ERREICHT WENN BEIDE
                        ;REGISTER GLEICH NULL SIND
                        ;UND DIE VORPROGRAMMIERTE
                        ;ZEIT UM IST
ENDZ:   OUT (PORTD), A  ;BITS UND AUSGABE AUF PORTD
        POP DE          ;REGENERIEREN DER IM
        POP BC          ;UNTERPROGRAMM ZEITSCH
        POP AF          ;BENUETZTEN REGISTER
                        ;FUER DAS HAUPTPROGRAMM
        RET             ;RUECKSPRUNG INS HAUPTPROGR
```

Dabei wird eine Leitung eines Z80-PIO als Zeitgeber verwendet; die Länge des Impulses an dessen Leitung ist durch die Summe der Befehlsausführungszeiten zwischen den Markierungen "ANF" und "ENDZ" gegeben, wobei der Befehl DEC C und der auf ihn folgende JR NZ, ZT1 m-mal durchlaufen wird, wobei m die durch den Befehl LD BC, ZEIT ins Register C geladene Zahl ist, und der Befehl DEC B und auf ihn folgende JR NZ, ZT1 n-mal, wobei n die durch den Befehl LD BC, ZEIT ins Register B geladene Zahl ist, während alle übrigen Befehle dieses Bereichs genau einmal ausgeführt werden.

Daraus folgt, daß mit unserem Programm ein Impuls am betreffenden PIO-Pin mit folgender Länge erzeugt wird:

$$= m \times n \times (\{\text{Bef. Ausführungszeit von LD rr, konst}\} + \{\text{Bef. Ausführzeit von DEC r}\} + \sum_{\text{übrige Befehle}} \text{Bef. Ausführzeiten}$$

Die einzelnen Befehlsausführungszeiten sind für die CPU-Nenn-Taktfrequenz dem Programmierkärtchen oder dem Programmierhandbuch zu entnehmen und auf die tatsächlich verwendete Taktfrequenz umzurechnen.
Wie man sieht, sind die Programme zur Realisierung von Mikrocomputer-Grundfunktionen einfach.
Sie haben jedoch folgende Nachteile:

- sie belasten die CPU mit einfachen Zählerfunktionen
- sie haben begrenzte Flexibilität

  (softwaregesteuerte Änderung erfordert zusätzlichen Programmieraufwand).
- sie stellen - wie logischerweise jeder einzelne Befehl eines Anwenderprogramms - bis zum endgültigen Austesten des Programms eine mögliche Fehlerquelle dar.

Es ist daher in Fällen, wo die CPU bereits durch andere Aktivitäten ausgelastet ist, sinnvoll, Zähl- und Zeitgeber-Vorgänge von speziellen Mikrocomputerperipheriebausteinen durchführen zu lassen, die diese Probleme unabhängig von der CPU und parallel zu ihrer sonstigen Arbeit erledigen bzw. sie nur zur Festlegung vorzuprogrammierender Parameter belasten.

Ein solcher Baustein ist beispielsweise der Z80-CTC, der vier Kanäle beinhaltet, die sich sowohl als Zähler, als auch als Zeitgeber betreiben lassen, wobei alle Funktionen (wie Interruptsteuerung, Zähler-Vorwahl und -Triggerung, Festlegung der auszugebenden Impulslänge usw.) durch Kommandoworte gesteuert werden, die per Anwenderprogramm einfach über Ausgabebefehle übergeben werden. Wir werden diesen Baustein in einem späteren Abschnitt noch genauer kennenlernen.

An dieser Stelle wird jedoch bereits ein Grundprinzip der Mikrocomputertechnik offensichtlich:

Praktisch sämtliche Funktionen eines Mikrocomputersystems können per Software (also durch Teile des Anwenderprogramms) oder per spezifischer Hardware (also durch spezielle Mikro-Computerperipheriebausteine oder durch eine zusätzliche Standardlogik-Schaltung) realisiert werden.

Aus diesem Grund sollte ein guter Mikrocomputerentwickler auf beiden Gebieten bewandert sein, in Hardware genauso wie in Software (= "Systemtechniker").

Beide Realisierungsformen haben grundsätzliche Vorteile:

- Die Realisierung einer Funktion in Software ermöglicht minimale Bauteilekosten und minimalen Platzaufwand.
- Die Realisierung einer Funktion in Zusatzhardware vereinfacht das Anwenderprogramm, hilft damit Softwareentwicklungskosten sparen und ermöglicht eine schnellere Verarbeitung bzw. einen höheren Durchsatz (="Throughput").

Welcher Lösungsform für das einzelne Teilproblem den Vorzug zu geben ist, hängt vom Einzelfall ab (geplante Gerätestückzahl, Geschwindkeitsanforderungen). Die Entscheidung liegt beim Geräteentwickler.

bei den Anforderungen die vom Anwender spezifiziert werden

2.6.6. Direkter Speicherzugriff

Alle bisher besprochenen Datenübertragungen zwischen Peripherie und dem Arbeitsspeicher (übrigens auch zwischen zwei Arbeitsspeicherbereichen!!) erfolgen nach dem Schema (Bild 13):

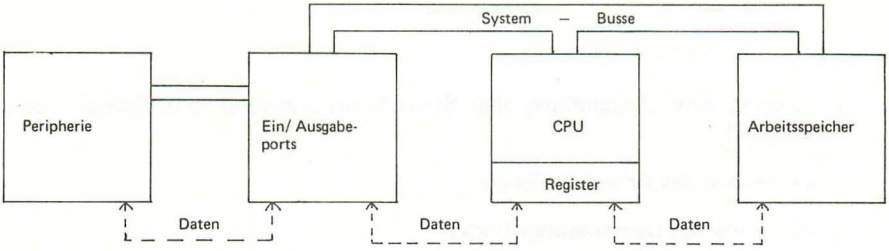


Bild 13: Datenübertragung ohne DMA

Eingabe

a) Übergabe eines Datenwortes von der Peripherie ins Eingabe-Port

b) Einlesen des Datenwortes in ein Arbeitsregister der CPU

c) Einschreiben des Datenwortes in die gewünschte Speicherzelle

Ausgabe

a) Lesen eines Datenwortes aus dem Arbeitsspeicher in ein Arbeitsregister der CPU

b) Ausgabe des Datenwortes in das Ausgabe-Port

c) Übernahme des Datenwortes durch die Peripherie vom Ausgabe-Port

In manchen Fällen - nämlich dann, wenn hohe Datenübertragungsraten erforderlich sind - ist dieses Verfahren zu zeitraubend.

Man muß in diesen Fällen eine Einrichtung schaffen, die eine Peripherie/Speicher- bzw. Speicher/Speicher-Übertragung **unter Umgehung** des Zentralprozessors ermöglicht (Bild 14). Dieses Verfahren heißt **direkter Speicherzugriff** (**D**irect **M**emory **A**ccess = DMA); es bedingt, wie man sich leicht überlegen kann, eine relativ hohe "Intelligenz" der Peripherie, die bei DMA die "Verwaltung" der Datenübertragung übernehmen muß, da der Zentralprozessor ja definitionsgemäß "umgangen" wird.

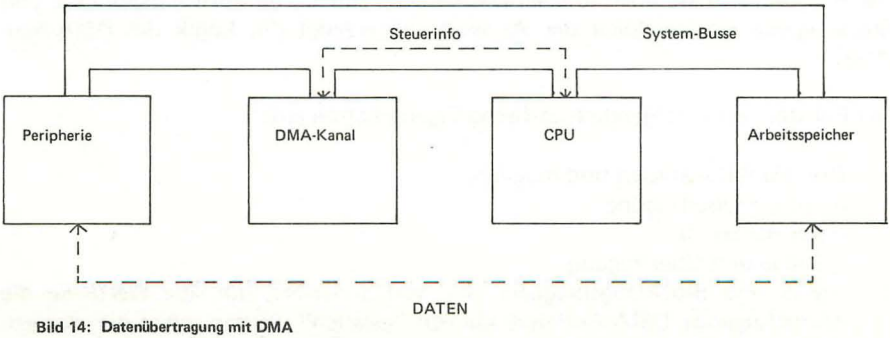


Bild 14: Datenübertragung mit DMA

Diese Aufgaben sind vor allem:

- Erzeugung bzw. Behandlung aller Steuerbusinformation (z.B. Schreib- oder Lesesignal),
- Adressierung des Arbeitsspeichers
- Feststellung des Übertragungs-Endes
- Interrupt-Behandlung
- Sperren bzw. Freigeben der CPU.

Wegen der hohen Komplexität dieser Aufgabe wird in der Mikrocomputertechnik hierfür ein spezieller Mikrocomputerbaustein verwendet, der auf den Zentralprozessor der entsprechenden Familie zugeschnitten ist. Er erledigt die vorher beschriebenen Aufgaben und darüberhinaus die Priorisierung von DMA-Anforderungen bei Vorhandensein mehrerer DMA-Kanäle, die alle auf den gleichen Speicherbereich zugreifen wollen.

Eine typische Konfiguration mit direktem Speicherzugriff zeigt Bild 14.
Auf den entsprechenden Baustein des Z80-Systems, den Z80-DMA, soll hier noch etwas näher eingegangen werden.

Beim Baustein Z80-DMA handelt es sich um einen hochintegrierten programmierbaren Baustein zur Implementierung "direkter Speicherzugriffe"

- ☐ für die besonders schnelle Übertragung von Daten (bis 1,25 Megabytes/sec) zwischen zwei Funktionalblöcken (="Ports") innerhalb eines Z80-Systems. Ports können hierbei sowohl Arbeitsspeicher als auch Ein/Ausgabe-Schnittstellen peripherer Geräte sein.
- ☐ zur schnellen Suche nach einem bestimmten Byte oder einer 8-Bit-Kombination (bit maskierbar) in einem Datenblock. Der Anwender kann zusätzlich festlegen, ob das gesuchte Zeichen gleichzeitig übertragen werden soll oder nicht.

Der Baustein enthält einen solchen DMA-Kanal. Die hierfür notwendigen Zeit- und Steuersignale einschließlich der Adressierung erzeugt die Logik des DMA-Bausteins.

Der Baustein weist folgende spezifische Eigenschaften auf:

- ☐ Drei Funktionstypen sind möglich:
  - o nur die Übertragung
  - o nur die Suche
  - o Suche und Übertragung
- ☐ Adreß- und Blocklängenregister sind **voll** gepuffert, d.h. die Werte für die nächstfolgende DMA-Aktivität können "geladen" werden, ohne die momentane Aktivität zu stören.

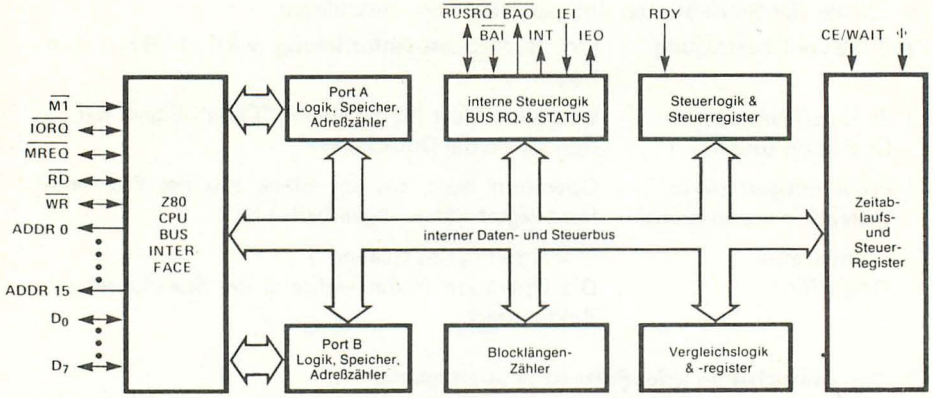
- ☐ Bei der Übertragung und Suche wird die Portadresse, beginnend von einer vorprogrammierbaren Anfangsadresse, automatisch inkrementiert bzw. dekrementiert.
- ☐ Seine vier Betriebsarten sind per Programm festzulegen:

☐ Ein-Byte-Übertragung	Pro Bus-request-Anforderung wird 1 Byte übertragen.
☐ Peripheriegesteuerte Operation (burst)	Operation läuft bis zu einem "Ende" Signal der Peripherie an die DMA-Ports.
☐ Programmgesteuerte Operation (continuous)	Operation läuft, bis ein Block mit per Programm festgelegter Länge abgearbeitet ist.
☐ transparente Operation:	("=refresh cycles stealing") Die Operation findet während der Speicherrefresh-Zyklen statt.

- ☐ Das Zeitverhalten jedes Ports ist programmierbar.
- ☐ Beim Auffinden eines gesuchten bytes oder bei Übertragungsende können Interrupts vorprogrammiert werden.
- ☐ Eine Operation kann automatisch oder auf Kommando vollständig wiederholt werden.
- ☐ Sobald eine bestimmte Anzahl von bytes übertragen worden ist kann vom DMA-Baustein ein Strobe-Signal ohne Unterbrechung der Übertragung abgegeben werden.
- ☐ Mehrfach-DMA-Strukturen mit unterschiedlicher Priorität können in einfacher Weise realisiert werden.
- ☐ Der DMA-Kanal kann per Programm freigegeben, gesperrt oder zurückgesetzt werden.
- ☐ Per Software kann auch der Status des DMA-Kanals vollständig abgefragt werden.

- ☐ Automatische Interrupt-Vektorerzeugung ohne zusätzlichen Schaltungsaufwand mit Prioritätskaskadierung der Bausteine (daisy chain priority interrupt logic). Die Bestätigung der Busanforderung ("BUSRQ") erfolgt hierbei ebenfalls automatisch.
- ☐ Alle Ein- und Ausgänge sind TTL-kompatibel.
- ☐ Von der CPU können die momentanen Werte von Portadreß-, Blocklängen oder Status-Registern gelesen werden. Die Auswahl des zu lesenden Registers erfolgt durch ein Maskierungs-Steuerwort.

Bild 15 zeigt das Blockschaltbild des Z80-DMA. Er besteht aus folgenden Funktionseinheiten:



- o Bus interfaces
- o Steuerlogik und zugehörige Register: Sämtliche Steuerparameter (wie z.B. Betriebsart oder Funktionstyp) werden in dieser Schaltung erkannt und zwischengespeichert.
- o Adreß-, Byte-Zähler und Pulsschaltung: Hier werden die Portadressen für Lese- und Schreiboperationen erzeugt und in-bzw. dekremiert. Ferner setzt die Byte-Zähl-Schaltung im Statusregister ein Flag, wenn das "nullte" Byte transferiert wird. Außerdem erzeugt die Pulsschaltung jedes Mal, wenn die niederwertigen 8 bit des byte-Zählers mit dem Inhalt des "Puls" -Registers übereinstimmen, einen Impuls.
- o Schaltung für die Steuerung des DMA-Zeitverhaltens Das Zeitverhalten bei Lese- und Schreibzugriffen wird für beide Ports per Programm festgelegt.
- o Vergleicher: In diesem Block werden durch Vorgabe einer Maske (mask byte) bestimmte bits des aktuellen Worts mit einem gesuchten byte (match byte) verglichen. Sobald das gesuchte byte während eines Übertragungsvorganges gefunden wurde, wird im Statusregister ein Flag gesetzt.

- o Interrupt- und BUSRQ-Schaltung: Im Interrupt Steuerregister wird festgelegt, unter welchen Bedingungen der DMA-Baustein einen Interrupt erzeugen darf. Ob dann zu diesem Zeitpunkt ein INT- oder BUSRQ-Signal ausgegeben wird, hängt von der jeweiligen Prioritätscodierungs-Logik ab. Ferner ist in diesem Schaltungs-

teil ein Interrupt-Vektor-Register enthalten, dessen Inhalt einen Teil des Zeigers einer Interrupt-Bedienroutine darstellt.

- Zustandsregister: In diesem Register sind die laufenden und für den Fortgang bestimmenden DMA-Status-Bedingungen gespeichert.

Sinnvollerweise sind entsprechend dem vorher Gesagten drei Funktionstypen möglich:

- nur die Übertragung
- nur die Suche und
- kombinierte Übertragung und Suche

Bei einem Übertragungsvorgang werden die Daten von einem Port gelesen und byteweise in das andere Port geschrieben. (Die beiden Ports werden als Port A und Port B bezeichnet). Die Ports können so programmiert werden, daß sie entweder dem Arbeitsspeicher oder peripheren Ein-/Ausgabe-Schnittstellen zugeordnet sind. Folgende Datenübertragungen sind d möglich:

- von einer peripheren Einheit zu einer anderen
- von einem Bereich im Arbeitsspeicher zu einem anderen
- von einer peripheren Einheit zum Arbeitsspeicher und umgekehrt

Bei einem Suchvorgang werden die Daten nur gelesen und byteweise mit zwei Register im DMA verglichen. In einem Register steht das für den Vergleich vorgegebene byte (match byte), im anderen (falls erforderlich) ein Masken-byte. Dadurch wird das Aufsuchen einzelner bits oder beliebiger Kombinationen möglich. Sobald das gesuchte 8bit-Wort gefunden ist, wird im DMA-Baustein ein Statusbit gesetzt. Je nachdem, wie der DMA-Baustein programmiert wurde, wird dann der Suchvorgang abgebrochen und/oder ein Interrupt erzeugt.
Bei kombinierten Übertragungs- und Suchvorgängen wird solange ein Datenblock übertragen, bis das gesuchte byte bzw. die gesuchte bit-Kombination gefunden wurde. Bei Übertragung kann dann ähnlich wie beim Suchvorgang abgebrochen und/oder ein Interrupt erzeugt werden.

Die Adressierung der DMA-Ports ist entweder fest oder aber ab einer vorgegebenen Startadresse sequentiell inkrementierend bzw. dekrementierend. Die Länge des zu übertragenden Blocks ist durch das vom Anwenderprogramm programmierte Blocklängenregister festgelegt.

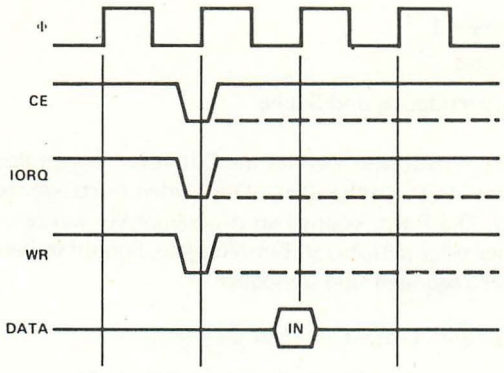
Der Z80-DMA kann Blocklängen bis zu 64 kBytes adressieren.

Das schematische Zeitverhalten des Bausteins ist wie folgt—

Zyklus zum Schreiben eines Kommandobytes

Ein Kommando- oder Steuerbyte wird in ein internes Z80-DMA-Register geschrieben. Das gezeigte Zeitverhalten ist durch die Z80-CPU Ausgabebefehle bestimmt.

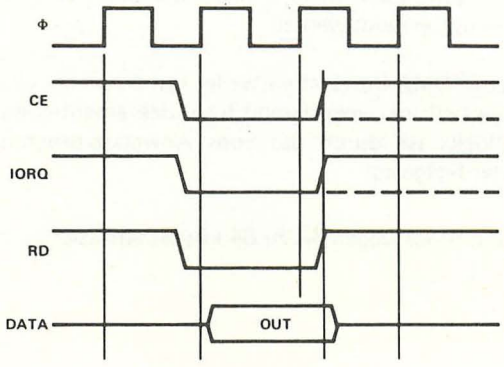
Bild 16: Schreibzyklus eines Kommandobytes



Zyklus zum Lesen eines Registers

Das Lesen des Inhalts des Statusregisters, des Adreßzählers oder anderer lesbarer Register Zähler geschieht nach folgendem Zeitverhalten, das durch die Z80-CPU-Eingabebefehle bestimmt ist.

Bild 17: Registerlesezyklus



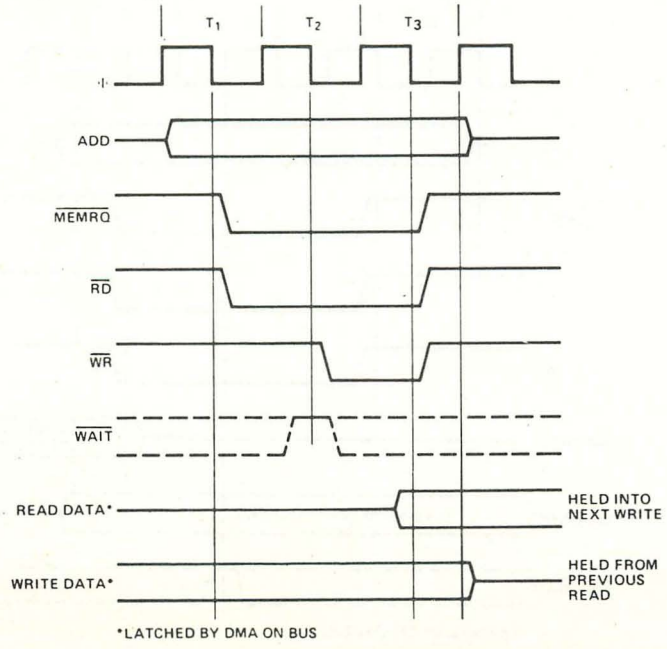
Speicherzugriff

Das Z80-DMA-Zeitverhalten beim Arbeitsspeicherzugriff entspricht sowohl beim Lese- als auch Schreibvorgang dem der Z80-CPU.
Dieses Zeitverhalten wird auch automatisch nach jedem RESET-Signal, beim Einschalten der Speisespannung und nach einem RESET-Kommando programmiert, und wird, falls kein anderes Kommandowort gesendet wird, bei allen Such- oder Übertragunsaktivitäten gegenüber dem Arbeitsspeicher beibehalten. Während der Speicherlesephase eines Übertragungszyklus werden die Daten im DMA mit der fallenden Taktflanke während T3 bis zu einem darauffolgenden Schreibzyklus zwischengespeichert. Während der Speicherschreibphase eines Übertragungszyklus werden die Daten vom vorausgegangenen Lesezyklus bis zum Ende des momentanen Zyklus zwischengespeichert.

Zur Beachtung:

Der DMA-Baustein ist für Speicherzugriffe mit 3 Maschinen (T-) Zyklen ausgelegt. Es ist jedoch das Einfügen von "$\overline{\text{WAIT}}$"-Zyklen möglich, da mit der abfallenden Taktflanke von T2 der log. Zustand des WAIT-Signals abgefragt wird. Ist dieser low, so wird ein weiterer T-Zyklus angehängt und das $\overline{\text{WAIT}}$-Signal nochmals abgefragt. Auf diese Weise kann die Dauer von Speicherzugriffen beliebig ausgedehnt und ein Anpassen an langsamere Speicher problemfrei vorgenommen werden.

Bild 18: DMA-Speicherzugriff



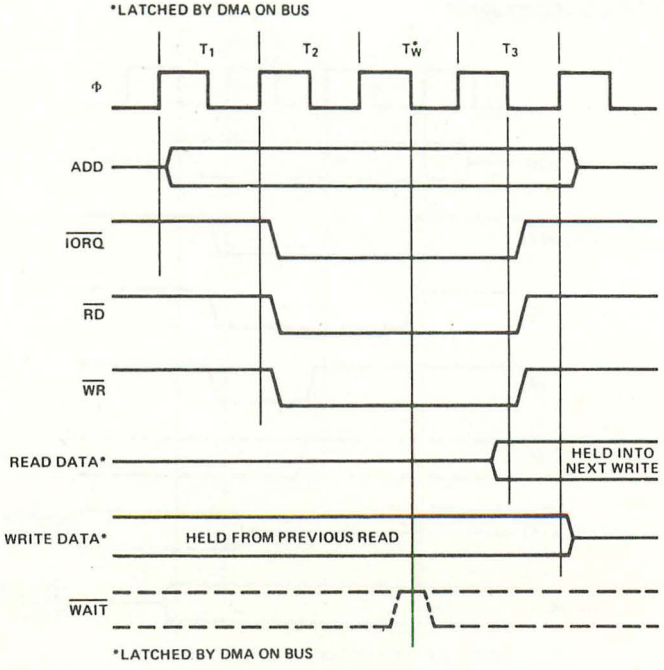
Verkehr mit Ein-/Ausgabeeinheiten

Das Zeitverhalten ist in diesem Fall identisch mit dem der Z80-CPU für 'Lese-Schreiboperationen mit Ein/Ausgabe-Bausteinen, und wird dem Baustein Z80-DMA beim Anlegen der Speisespannung, einem Signal auf der RESET-Leitung oder nach einem Reset-Kommandowort automatisch vorgegeben. Es wird, falls dem Z80-DMA keine anderen Kommandowörter zugesendet werden, bei allen Such- und Übertragungsvorgängen mit peripheren Ein-/Ausgabe-Schnittstellen beibehalten. Während der Ein-/Ausgabe-Lesephase eines Übertragungszyklus werden die Daten mit der abfallenden Taktflanke des T3-Taktimpulses bis zum nächstfolgenden Schreibzyklus zwischengespeichert. Während der Ein-/Ausgabe-Schreibphase eines Übertragungszyklus werden die Daten vom vorangegangenen Lesezyklus bis zum Ende des momentanen Zyklus zwischengespeichert.

Zur Beachtung:

Ist das WAIT-Signal während der fallenden Taktflanke von Tw* gleich low, dann wird ein neuer T-Zyklus angehängt und das WAIT-Signal nochmals abgefragt. Auf diese Weise kann die Dauer von peripheren Ein-/Ausgabezugriffen beliebig ausgedehnt werden.

Bild 19: Verkehr mit Ein-/Ausgabeeinheiten



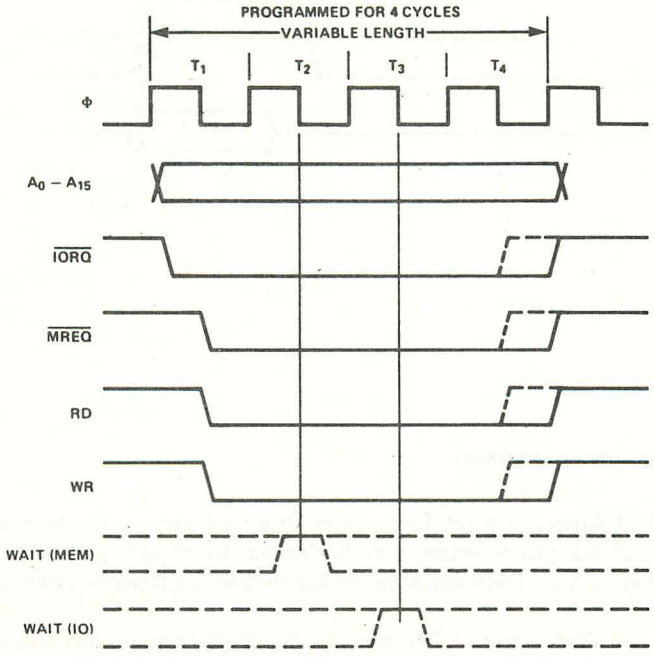
Variabler Zyklus

Eine besondere Eigenschaft des Z80-DMA ist seine Möglichkeit, das Zeitverhalten von Speicher oder Ein/Ausgabe-Vorgängen frei zu prgrammieren. Damit kann

a) das Zeitverhalten an die jeweiligen Anforderungen der gewählten Systemkomponenten angepaßt und
b) die Datentransferrate optimiert werden,

ohne daß hierfür zusätzliche Schaltungen oder Programmteile nötig werden. Die Zykluslänge kann auf ein bis 4 Maschinen-(T-)Zyklen (und mehr bei Verwendung von WAIT-Zyklen) vorprogrammiert werden. Die Signallängen können entsprechend nebenstehendem Impulsdiagramm variiert werden. Die Datenzwischenspeicherung während des Übertragungsvorganges erfolgt nach der Taktflanke, die eine steigende Flanke von RD- zur Folge hat. Bis zum Ende des folgenden Schreibzyklus werden diese Daten gespeichert.

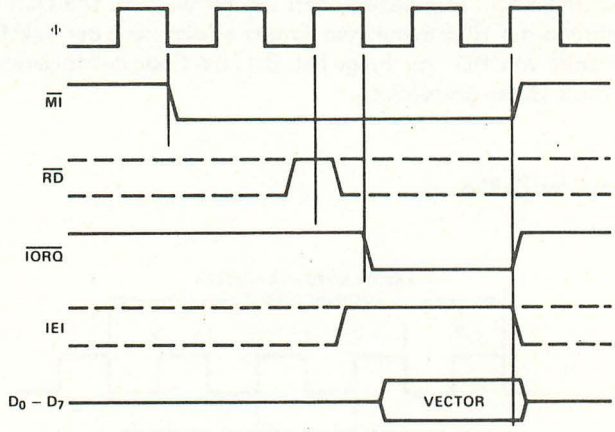
Bild 20: Variabler Zyklus



$\overline{M1} \wedge \overline{ORQ}$ Interrupt-Quittungszyklus

Nachdem vom Z80-DMA ein Interrupt angefordert wurde, wird von der Z80-CPU eine Interrupt-Quittung ausgegeben ($\overline{M1} \wedge \overline{IORQ}$). In der Zwischenzeit wird durch die Interruptlogik festgestellt, ob der DMA-Kanal die höchste Priorität hat. (Die Interruptkaskade muß dazu erst einschwingen. Es dürfen sich deshalb die Zustände der Interruptanforderungen nicht ändern, solange $\overline{M1}$ aktiv ist). Ein DMA-Vektor wird dann auf den Datenbus ausgegeben, sobald $\overline{IORQ}$ aktiv wird.

Bild 21: Interrupt-Bestätigungszyklus



Rücksprung vom Interruptzyklus

Durch eine RETI-Anweisung am Ende einer Interrupt-Service-Routine wird der eindeutige Ablauf bei priorisierten, verschachtelten Interrupts gewährleistet. Die RETI-Anweisung wird im DMA-Baustein zu Identifizierungszwecken dekodiert.

Bei Erkennen des Codewortes EDH wird das IEI-Signal des jeweils angesprochenen Bausteines aktiv. Wird 4 DH empfangen, so wird der momentan bediente Kanal neu initialisiert und sein IEO-Ausgang aktiviert.

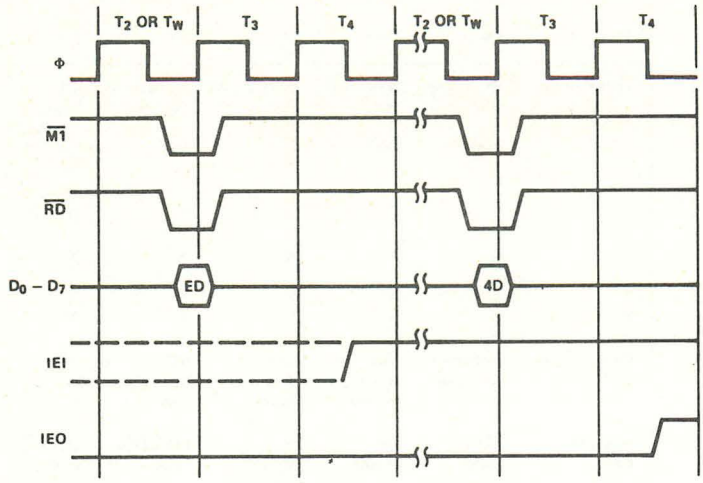


Bild 22: Rücksprungzyklus

Es soll hier kurz der Vorgang des Programmierens des DMA-Bausteins erläutert werden.

Im aktiven Zustand übernimmt der DMA-Baustein die Kontrolle über die Systembusse, den Datentransfer und die Interruptverarbeitung.
Ein Kommandowort kann dem DMA-Baustein sowohl im aktiven als auch im inaktiven Zustand übermittelt werden; es bewirkt automatisch einen Übergang in den nicht aktiven Zustand. In diesem Zustand befindet sich der Baustein auch, wenn die Stromversorgung eingeschaltet oder der DMA-Baustein auf eine andere Weise zurückgesetzt wird.
Bevor der DMA-Baustein seine eigentliche Aufgabe erfüllen kann, muß er durch das Anwenderprogramm über die CPU wie ein Ein-/Ausgabe Port initialisiert werden. D.h. durch Ausgabebefehle wird eine Folge von Kommandoworte über den Datenbus in die verschiedenen Z80-DMA-Register geladen. Die Kommandoworte enthalten 8bit-Steuerinformationen, die den Zustand und die Arbeitsweise des DMA-Bausteins bestimmen und beeinflussen wie z.B. das Setzen eines Interruptfreigabebits oder zusätzlich benötigte Informationen, die nach dem jeweiligen Kommandobyte geladen werden wie z.B. die Startadresse eines Ports.
Die Funktion jedes einzelnen bits der 6 verschiedenen Kommandobytes wird anhand folgender Zussammenstellung klar:

Die Kommandobytes sind in 2 Gruppen eingeteilt

Gruppe 1 mit den bytes 1 A, 1 B
Gruppe 2 mit den bytes 2 A, 2 B, 2 C, 2 D

und sind durch die bits D_0, D_1, D_7 gekennzeichnet.

Byte-Wort \ bit		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Commando-byte 1A		Gruppenspez. ↓ = „0"	Blocklänge höw. folgt	Blocklänge niw. folgt	Startadresse Port A (höw.) folgt	Startadresse Port A (niw.) folgt	Übertragungs-richtung „1" Port A → B „0" Port B → A	Funktionstypenschlüssel C1 ≠ „00"	C0
Startadresse Port A (niw.)	0	X	X	X	X	X	X	X	X
Startadresse Port A (höw.)	0	X	X	X	X	X	X	X	X
BL (niw.)	0	X	X	X	X	X	X	X	X
BL (höw.)	0	X	X	X	X	X	X	X	X
Commando-byte 1B		Gruppenspez. = „0"	Zeitsteuer-byte folgt	feste Adres-sierung	inkr. = „1" dekr. = „0" Adres-sierung	I/O = „0" Spei-cher = „1"	Port A = „0" B = „1"	Bytespezifikation = „0"	= „0"
Zeitsteuerbyte	0	$\overline{WR}$ (1) Ende	$\overline{RD}$ (1) Ende	nicht benützt	nicht benützt	$\overline{MREQ}$ (1) Ende	$\overline{IORQ}$ (1) Ende	Zykluslängenschlüssel T_1	T_0
Commando-byte 2A		Gruppenspez. = „1"	Chip Freigabe	Interrupt-Freigabe	Mask byte folgt	match byte folgt	Stop bei Vergleich	Bytesspezifikation „0"	„0"
match byte	0	X	X	X	X	X	X	X	X
mask byte („1" = Reg. wird gelesen)	0	nicht benützt	Adresse Port B höw.	Adresse Port B niw.	Adresse Port A höw.	Adresse Port B niw.	BL höw. Zähler	BL niw. Zähler	Status
Commando-byte 2B		Gruppen-spez. = „1"	Betriebsartenschlüssel M1	M2	Interrupt-steuer-byte folgt	Adresse Port B höw. folgt	Adresse Port B niw. folgt	Bytespezifikation = „0"	= „1"
Adresse Port B niw.	0	X	X	X	X	X	X	X	X
Adresse Port B höw.	0	X	X	X	X	X	X	X	X
Interrupt Steuerbyte (Ja = „1")	0	nicht benützt	Interrupt bevor Busanf.	Einfluß Status auf Interrupt-vektor	Interrupt-vektor folgt	Pulsanzahl folgt	Pulser zeugung wenn gesuchtes Byte gefunden Interruptstatus-schlüssel D2	Inter-rupt, D1	Inter-rupt am Block-ende
Pulsanzahl	0	X	X	X	X	X	X	X	X
Interruptvektor	0	X	X	X	X	X	X	X	X
Commando-byte 2C		Gruppen-spez. = „1"	nicht benützt	Auto restart	$\overline{CE}$ = „0" $\overline{CE}$ und $\overline{Wait}$ = „1"	Ready aktiv Signal high = „1" low = „0"	nicht benützt	Bytesspezifik. = „1"	= „0"
Commando-byte 2D		Gruppen-spez. = „1"	Zustandsschlüssel f_4	f_3	f_2	f_1	f_0	Bytesspezifik. = „1"	= „0"

Legende: o: optional
höw.: höherwertig
niw.: niederwertig
BL: Blocklänge

Funktionstypenschlüssel

C_1C_0	Funktion
0 0	nicht erlaubt
0 1	nur die Suche
1 0	nur die Übertragung
1 1	Suche und Übertragung

Betriebsartenschlüssel

M_1M_0	Ablaufweise
0 0	byteweise
0 1	fortlaufend
1 0	burst
1 1	transparent

Zykluslängenschlüssel

T_1T_2	Zykluslänge
0 0	1
0 1	2
1 0	3
1 1	4

(1) Eine „1" in D_2, D_3, D_6 oder D_7 hat zur Folge, daß das entsprechende Steuersignal eine halbe Taktzeit vor dem Zyklus beendet ist.

Interruptstatusschlüssel

(falls bit 5 des Interruptsteuerbytes gesetzt („1") wurde)

D_2D_1	Funktion
0 0	Ready aktiv
0 1	Vergleichsbyte gefunden, Blocklängenende nicht gefunden
1 0	Vergleichsbyte nicht gefunden, Blocklängenende gefunden
1 1	Ready nicht aktiv

Zustandsschlüssel

Hex	f_4	f_3	f_2	f_1	f_0	Bedeutung	Kurzbeschreibung
C3	1	0	0	0	0	Chip zurücksetzen	Setzt alle Interruptbedingungen zurück, sperrt Interrupts und Busse
C7	1	0	0	0	1	Setzt Zeitverhalten der Ports A/B zurück	gleicht das Zeitverhalten der Ports A/B an das Zeitverhalten der Z80-CPU an
CB	1	0	0	1	0		
CF	1	0	0	1	1	Restart	lädt nochmals Blocklänge und Startadresse für beide Ports
D3	1	0	1	0	0	Restart bei momentanem Adreßstand	lädt nochmals die Blocklänge. Die Ausführung wird beim momentanen Adreßstart begonnen.
AB	0	1	0	1	0	Interruptfreigabe	Interrupts werden erlaubt
AF	0	1	0	1	1	Interruptsperre	Interrupts werden nicht erlaubt
A3	0	1	0	0	0	Interrupt-rücksetzung	Alle Interrupts werden zurückgesetzt (vergleiche mit RETI)
87	0	0	0	0	1	Chip Freigabe	Alle Operationen mit Ausnahme des Interrupts werden freigegeben oder
83	0	0	0	0	0	Chip Sperre	blockiert; es wird nichts zurückgesetzt
BB	0	1	1	1	0	Lesemarke folgt	welche Register ausgegeben werden, steht im nächst folgenden byte
A7	0	1	0	0	1	Zurücksetzen von Read	Das erste auslesbare, durch die Lesemarke bestimmte Register wird als nächstes gelesen. Die Kommandos (Reset chip, Reset Timming, Restart, Continue, Enable Chip, Disable Chip, Response, Mask follows bewirken denselben Rücksetzvorgang.
BF	0	1	1	1	1	Lesen des Status (RD Status)	Als nächstes wird das Status Register ausgelesen
B3	0	1	1	0	0	Erzwungenes Ready-Signal (Force Ready)	Bei gewissen Operationen wird kein Ready-Signal benötigt (z.B. bei der Speicher zu Speicherübertragung). Das Ready-Signal wird hier als aktiv betrachtet, unabhängig vom tatsächlichen log. Zustand.
B7	0	1	1	0	1	Freigabe der Busanforderung nach einem RETI-Signal	Vom DMA-Baustein kommt keine Busanforderung, solange des RETI-Signal nicht empfangen wurde.
8B	0	0	0	1	0	Zurücksetzen des Status (RST Status)	Das Match und Blockende-Statusbit wird zurückgesetzt.

Status-Byte

nicht benützt	nicht benützt	Block-längenende	Match	steht ein Interrupt bevor	Ready aktiv	nicht benützt	gültige Adresse
D7	D6	D5	D4	D3	D2	D1	D0

Masken-Byte

Ein Vergleich zwischen dem gelesenen und dem vorgegebenen Bit in einem Datenwort findet statt, falls an der entsprechenden Stelle eine log "0" im Masken byte vorgegeben wurde.

Match-Byte

Wörter bis zu 8-bit (Do-D7) können während des Lesevorganges verglichen werden. Näheres siehe Masken-Byte.

Programmbeispiel

In dem unten gezeigten Beispiel sollen Daten (Blocklänge 1000 H Bytes) von einem peripheren Gerät (Port A) Adresse 05 H in den Arbeitsspeicher (Port B, Startadresse 1050 H) übertragen werden. Zur Initialisierung mindestens erforderlich sind nach dem Einschalten der Versorgungsspannung folgende Kommandowörter:

byte-Wort \ bit	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Commandobyte 1A	Gruppenspez.	Blocklänge höw	Blocklänge niw	Startadresse Port A höw	Startadresse Port A→B	Übertrag.-richtung Port A→B	Funktions-typenschlüssel hier: Suche und Übertragung	
	0	1	1	0	1	1	1	1
Startadresse Port A (niw), 05H	0	0	0	0	0	1	0	1
BL (niw)	0	0	0	0	0	0	0	0
BL (höw)	0	0	0	1	0	0	0	0
Commandobyte 1B	Gruppenspez.	Zeitsteuer-byte folgt	feste Adres-sierung	Adressie-rung inkre.	Speicher	Port A	byte-Spezi-fikation	
(Def. Port A)	0	0	1	X	1	0	0	0
(Def. Port B)	0	0	0	1	0	1	0	0
Commandobyte 2B	Gruppenspez.	Betriebsartenschlüssel hier: burst		Interrupt Steuerbyte folgt	Adresse Port B höw folgt	Adresse Port B niw folgt	byte-Spezi-fikation	
	1	1	0	0	1	1	0	1
Startadresse Port B niw	0	1	0	1	0	0	0	0
Startadresse Port B höw	0	0	0	1	0	0	0	0
Commandobyte 2C	Gruppenspez.	nicht benutzt	Autore-start	CE	Ready aktiv high	nicht benutzt	byte-Spezi-fikation	
	1	X	0	0	1	X	1	0
Commandobyte 2D	Gruppenspez.	f_4	f_3	f_2	f_1	f_0	byte-Spezi-fikation	
	1	0	0	0	1	0	1	1

2.6.7. Periphere Hardware für 'Mikros"

Da sich mit Mikroprozessorbausteinen ebensogut Gerätesteuerungen wie "General Purpose"-Computer realisieren lassen, kommt praktisch jede beliebige Computerperipherie zur Verwendung in Mikrocomputersystemen infrage. In dieser Arbeit sollen allerdings nur die typischsten Klassen solcher Geäte herausgegriffen und dem Nicht-Computertechniker vorgestellt werden.

Periphere Einheiten sind in der Regel mit der Zentraleinheit über "PORTS" (=Kanäle) verbunden und erhalten von diesen ihre Arbeitsanweisungen nach Maßgabe eines eigenen Unterprogramms (das sogenannte "Treiberprogramm"), das speziell auf die angeschlossene periphere Einheit zugeschnitten ist. Die Aufgaben peripherer Einheiten sind:

- o Eingabe von Daten
- o Ausgabe von Daten
- o Speicherung von Daten und
- o Kommunikation zwischen Zentraleinheit und Benutzer bei Computern mit Betriebssystemen (OEM-, General Purpose- oder Mikrocomputerentwicklungs-Systemen).

Aufgrund dieser Aufgabenstellung können periphere Einheiten eingeteilt werden in:

- o Eingabeeinheiten
- o Ausgabeeinheiten
- o Externe Speichereinheiten
- o Dialogeinheiten

a) Eingabeeinheiten dienen, wie der Name sagt, in erster Linie zur Eingabe (Lesen) von Daten in die CPU-Register oder in den Schreib/Lese-Speicher der Zentraleinheit. Da die Daten nicht immer in dem von der Zentraleinheit verwendeten internen Verarbeitungs-Code auf den einzelnen Datenträgern dargestellt sind, wird häufig eine Code-Umwandlung durchgeführt.

   Die am weitesten verbreiteten Eingabegeräte sind:

   - o Tastatur
   - o Lochstreifenleser
   - o Gerätekontakte (bzw. deren opto- oder magnetoelektronischer Ersatz)
   - o Analog/Digital-Wandler
   - o Codierschalter

b) Ausgabeeinheiten dienen entsprechend zur Ausgabe der von der Zentraleinheit ermittelten Ergebnisdaten, je nach Verwendungszweck, in eine für den Menschen bzw. die Maschine verständliche Form infrage kommen:

- o Matrixdrucker
- o Zeilendrucker
- o Leuchtdioden-Display (z.B. Sieben-Segment-Anzeige)
- o Leuchtfelder (beleuchtetes Schriftfeld)
- o alphanumerische/graphische Bildschirmeinheit (z.B. Heim-Fernseher)
- o Lochstreifenstanzer
- o Plotter (Zeichengerät)
- o D/A-Wandler
- o Relais/Triacs/Leistungstransistoren
- o Sämtliche denkbare Interfaces zu Servo-Zwecken

c) Externe Speichereinheiten.
Bei vielen Anwendungen genügt die an den Mikroprozessor anschließbare Schreib/Lesespeicherkapazität nicht, oder aber die entsprechenden Kosten an MOS-Schreib/Lesespeicherbausteinen sind zu hoch. In diesen Fällen verwendet man externe (Massen-) Speichereinheiten, die wesentlich niedrigere Kosten pro gespeichertem Bit verursachen und darüber hinaus folgende Vorteile haben:

- Die Daten sind auf dem Speichermedium in nichtflüchtiger Form aufgebracht, d.h. sie gehen - anders als beim üblichen MOS-Schreib/Lesespeicherbaustein - bei Ausfall oder Abschalten der Versorgungsspannung nicht verloren.
- Der Datenträger läßt sich aus der Speichereinheit herausnehmen und transportieren (z.B. zu einem anderen Standort schicken).
- Durch Auswechseln der Datenträger steht dem Computer eine praktisch unendlich große Informationsmenge zur Verfügung.

Da auch auf diesem Sektor die Preise durch den technologischen Fortschritt dauernd sinken, erfreuen sich solche Speichermedien dauernd wachsender Beliebtheit. Für Mikrocomputersysteme werden heute in der Regel folgende Massenspeichereinheiten eingesetzt:

- Floppy Disks
- Minifloppies
- Magnetplattenspeicher
- Magnetbandkassetten-Geräte
- Magnetbandgeräte (9-spurig, IBM-kompatibel)

Auf die Geräte werden wir im einzelnen noch zurückkommen.

Hier noch ein Überblick über

d) Dialogeinheiten:
Man benutzt sie zur Abwicklung des Mensch/Maschinekontakts in Fällen, wo ein "Dialog" zwischen Bediener und Mikrocomputer erforderlich ist. Dies ist immer der Fall bei OEM-Computern (z.B. ZILOG MCS) und Mikrocomputer-Entwicklungssystemen, wo damit der Verkehr mit dem Betriebssystem abgewickelt wird, aber auch grundsätzlich bei Anwendungen, wo der Benutzer ak-

tiv festlegt, welche Informationen er bekommen möchte, statt auf das "Informationsangebot" des Computers zu warten (z.B. interaktiver Arbeitsplatz in Prozeßsteuerung).

Solche Einheiten bestehen aus einer Eingabetastatur und einer Ausgabemöglichkeit; beides ist meist über eine Serienschnittstelle mit der CPU verbunden. Hierfür stehen grundsätzlich zur Verfügung:

- Druckende Dialogeinheiten (z.B. Fernschreiber, Thermodrucker)
- Nichtdruckende Dialogeinheiten (Datensichtgeräte, alphanumerisch und/oder grafische Darstellung)

Denkbar wäre auch der direkte Kontakt über die menschliche Sprache. Entsprechende Einheiten befinden sich allerdings noch in einem frühen Entwicklungsstadium (= "Vococoder").

2.6.8. Wahlfreier Zugriff auf Massenspeicher durch Magnetplattentechnik

Benutzt man als externe Daten- oder Programmspeicher die bereits besprochenen Magnetband-, Magnetbandkassetten- oder Lochstreifengeräte, so muß man bei jedem Zugriff auf die gesuchte Information den ganzen Datenträger durchlaufen lassen, bis man die Stelle findet, an der die gesuchte Information gespeichert ist.

Das kostet Zeit, und da in der Computertechnik mit Nanosekunden gerechnet wird, ist dieses Verfahren in vielen Fällen nicht tragbar.

Die Gerätetechniker haben sich deshalb ein raffinierteres Verfahren einfallen lassen, das dem Prinzip des bekannten Plattenspielers ähnelt, jedoch im Gegensatz zu diesem den Lesekopf nicht in Form einer Spirale vom Plattenrand zur Plattenmitte führt, sondern die Platte auf konzentrischen Spuren abtasten läßt.
Mit diesem Verfahren erhält man im Speichermedium, den "WAHLFREIEN ZUGRIFF" erlaubt, d.h. das Lesen einer Datei ermöglicht, ohne dabei den ganzen Datenträger auf den Anfang dieser Datei durchsuchen zu müssen.

Man verwendet dabei einen scheibenförmigen gleichmäßig rotierenden Datenträger, der mit einer magnetisierbaren Schicht (ähnlich wie das Tonband) beschichtet ist und einen Schreib/Lesekopf, der von einem Schrittmotor in eine bestimmte Position gebracht wird, in der er auf eine Spur zugreift.
Die so definierten, kreisförmigen konzentrischen Spuren sind nochmals in Sektoren unterteilt, die jeweils datentechnische Einheiten (= "Datensätze" oder "Records") enthalten.
Wir wollen hier nicht in die letzten Einzelheiten der kommerziellen Datentechnik oder der Magnetplattentechnik einsteigen, es sollen jedoch folgende Grundprinzipien vermittelt werden:
Unter einer Datei versteht man eine beliebig umfangreiche Informationsmenge. Man legt eine Datei im allgemeinen in einem zusammenhängen Bereich der Speicherkapazität eines Speichermediums ab.

Jede Datei enthält neben den eigentlichen Informationen Angaben über Datei-Namen, -Eigenschaften und -Länge, außerdem eventuell über die Adresse der folgenden Datein u.s.w.
Den Begriff einer **Datei** machen Sie sich am besten an der entsprechenden Bedeutung einer **Kartei** klar. Auch hier haben Sie eine geschlossene Datenmenge die aus Untereinheiten (= "Karten") besteht.
Analog hierin besteht eine Datei i.a. aus den oben erwähnten **"Sätzen"** (= "Records"), die feste (z.B 128Byte) oder variable Länge haben können.
Die Informationseinheit ist dabei das Byte.
Die Speicherung eines jeden Bytes erfolgt in bit-serieller Form.
Die technologische Entwicklung hat zu folgenden Produktgruppen geführt:

- Festkopfplatte: Für superschnellen Zugriff ohne die Notwendigkeit oder Möglichkeit der Positionierung.
- Magnetplatte (= "Hard Disk") : Plattenspeicher, dessen Datenträger aus einer oder mehreren Magnetplatten ("Plattenstapel") besteht, der von einen Schreib/Lese-Kopf abgetastet wird, der auf einem Luftpolster läuft.
- Floppy - Disk : Billigst-Plattenspeicher, dessen Datenträger **eine** magnetisch beschichtete Scheibe ist, die in einer Papierhülle rotiert.
  Durch Reibungseffekte und die Tatsache, daß der Schreib/Lesekopf mechanischen Kontakt mit dem Datenträger hat, ist die Lebensdauer der "Diskette" (Datenträger der Floppy Disk) nicht unbegrenzt.
  Da man den Zeitpunkt der Zerstörung der Information auf dem Datenträger nicht mit genügender Sicherheit vorhersehen kann, ist es sinnvoll, laufend grundsätzlich Kopien jeder Datei auf einer zweiten Diskette anzufertigen.

Grundsätzlich sind in der Magnetplattenspeichertechnik folgende Adressierweisen möglich:

- Sequentielle Verarbeitung
- die gestreute Speicherung mit direkter Adressierung
- die index-sequentielle Speicherung

Die **gestreute Speicherung** mit **direkter Adressierung** gestattet eine **wahlfreie Verarbeitung** jedes Satzes einer Datei.

- o Neue Sätze werden in den entsprechenden Lücken im Speicherbereit der Datei eingefügt
- o Es müssen nur die interessierenden Sätze gelesen werden
- o Durch Vorsortierung der interessieren Sätze kann die Verarbeitungszeit verkürzt werden

Dateien, deren Ordnungsbegriffe sich nicht für eine direkte Adressierung eignen, können mittels der **gestreuten Speicherung** mit **indirekter Adressierung** gespeichert und wahlfrei verarbeitet werden.

- o Neue Sätze werden in Lücken oder einem Folgebereich eingefügt.
  Es kann eine eigene **Überlaufspur** oder aber ein gemeinsamer, unabhängiger **Überlaufbereich** vorgesehen werden.
- o Bei der Verarbeitung werden nur die interessierenden Sätze angesprochen.

Die **index-sequentielle (logisch fortlaufende) Speicherungsform** eignet sich ebenfalls gut für die wahlfreie Verarbeitung.

- o Beim Suchen eines bestimmten Satzes muß nur der **Adreßindex** und nicht der gesamte Datenbestand gelesen werden.
- o Neue Sätze werden entsprechend der Sortierfolge zwischen den bereits gespeicherten Sätzen eingefügt.
  Die dem neuen Satz folgenden Sätze der Spur müssen hierzu um einen Satz nach rechts verschoben werden, so daß der letzte Satz der Spur herausfällt.
- o Der herausfallende Satz wird als Folgesatz der betreffenden Spur in einen **Folgebereich** gespeichert. Die Eintragungen im **Spurindex** ermöglichen den Zugriff zu den ausgelagerten Sätzen.
- o Durch sortierte Eingabe der interessierenden Sätze kann die Verarbeitungszeit reduziert werden.

Welche Adressierweise die günstigste ist hängt vom jeweiligen Anwendungsfall und der verfügbaren Steuer-Software ab.

In der Mikrocomputertechnik werden überwiegend Floppy-Disks eingesetzt. Ihre Anwendbarkeit erweitert sich durch die sinkenden Preise der FD-Laufwerke und der Schreib-Lesespeicherbausteine dauernd.

Dadurch wird möglich:

- mehrere Programme größeren Umfangs für ein und dasselbe System zu entwikkeln und auf einer relativ billigen Hardware ablaufen zu lassen, ohne umfangreiche Festwertspeicherbereiche realisieren zu müssen.

- die Flexibilität von Mikrocomputersystemen dahingehend zu erhöhen, daß man zur Änderung der Systemfunktion nicht einmal mehr den Programmspeicher (PROM), sondern lediglich einen Datenträger (Diskette) austauscht.

Bei den Floppy-Disks haben sich zwei Standards herauskristallisiert:

- Die "Softsectored Floppy" (auch "IBM-kompatibel" genannt), bei der lediglich der Sektor Null hardwaremäßig von der Leseeinrichtung detektiert wird (nämlich durch ein einziges Loch im Datenträger).
- Die "Hardsectored Floppy", bei der die Diskette 32 Sektorlöcher aufweist, die hardwaremäßig jeden einzelnen Sektoranfang definieren.

Dieses Verfahren erlaubt eine bessere Ausnutzung des Datenträgers und damit eine größere speicherbare Datenmenge/Diskette und eine einfachere softwaremäßige Behandlung des Gerätes, erfordert jedoch zusätzliche Hardwaremaßnahmen im Flopp-Disks-Controller.

Man erreicht mit diesem Verfahren 0.25...0.3 MioByte effektive Speicherkapazität pro Diskette.

Die Entwicklung der Floppy-Disk-Technik hat inzwischen zu folgenden Variationen geführt:

- Floppy-Disk mit doppelter Schreibdichte (= "double-density"); dadurch wird eine Verdoppelung der genannten Speicherkapazität pro Diskette erreicht.
- Floppy-Disk mit doppelseitiger Speicherung ("Double-Sided"), wodurch eine nochmalige Verdoppelung der Speicher-Kapazität pro Diskette erzielt wird.
- Mini-Floppy, die wenig mehr als den halben Durchmesser der Standard-Diskette (20 cm) hat. Ihre Speicherkapazität ist 1/8 MioByte pro Minidiskette in der Standardausführung (also bei einfacher Schreibdichte, einseitig beschrieben).

2.6.9. In Sache Mikrocomputer - Software

2.6.9.1. Wahl der Programmiersprache

Einer der wesentlichsten Unterschiede zwischen der Mikrocomputer- und der Mini- bzw. Großrechner-Technik ist die Tatsache, daß bei den "Mikros" insbesondere beim Verkehr mit der "realen" Außenwelt eine starke Hardware-Gebundenheit besteht.
Dies ist unvermeidlich, da der Hauptanwendungszweck von Mikrocomputern ja gerade der ist, sehr spezifische Peripherie zu steuern (z.B. eine Drehbank, eine Motorzündung in einem KFZ oder eine Kaffeemaschine) und nicht ein Standardgerät wie irgendeinen Drucker oder Fernschreiber, wie das in der übrigen Rechnertechnik ausreichend ist.
Durch diese Hardware-Bindung ist auch eine starke Bindung an die Rechnerarchitektur impliziert, was wiederum bewirkt, daß mindestens an diesen Stellen rechnernah-, also in Assemblersprache programmiert werden muß, d.h. diese Bereiche bleiben bekannten High-Level-Computersprachen wie FORTRAN, COBOL, ALGOL, PL/I, PASCAL oder BASIC verschlossen.

Die einzige dem Autor bekannte Mikrocomputerspezifische Aktivität in der Richtung, Mikrocomputer auch auf dieser Ebene anwenderfreundlich zu programmieren, ist ZILOG's PL/Z.

Dies ist eine Strukturierungssprache, die eine virtuell 100%ige Effizienz bezüglich Speicherbedarf und Verarbeitungsgeschwindigkeit gegenüber ASSEMBLER ermöglicht; sie benützt zwar ähnliche Sprachelemente wie PL/M, hat jedoch ansonsten mit dieser Sprache und anderen PL/I-Derivaten herzlich wenig zu tun.

Stattdessen versucht sie mit Strukturierungselementen (wie DO....WHILE-Schleifen oder PROCEDURE;) dem Programmierer die Sorge um CPU-Register und Adressierungabzunehmen, ohne dabei Maschinencode zu erzeugen, der statisch oder dynamisch ungünstiger ist als die entsprechende Codierung in ASSEMBLER.

Für Fälle der technisch-wissenschaftlichen und der kommerziellen Datenverarbeitung oder einfach zur raschen Datenauswertung im Labor sind diese Gesichtspunkte jedoch nicht maßgeblich.

Hierfür sind nach wie vor die klassischen Programmiersprachen interessant.
An der Spitze davon steht die Sprache BASIC, eine Dialogsprache, die die interaktive Behandlung von Problemen ermöglicht, der Computer "hilft" hierbei tatsächlich beim Auffinden der Lösung, algebraische Probleme werden ähnlich wie beim Taschen- oder Tischrechner durch Standard-Befehle erledigt.
Im Falle des Z80-BASIC ist es über den üblichen BASIC-Rahmen hinaus möglich, sich auf einfachste Weise Dateien auf Diskette zu erzeugen, die dann von anderen Programmen weiterverarbeitet werden können, oder aber sogar ASSEMBLER-Programme "einzubinden" die direkt mit Prozeßperipherie "sprechen" können.

Da die Mikrocomputer durch ihre immer wachsende Leistungsfähigkeit immer weiter in den kommerziellen Bereich vorstoßen, gibt es für das System Z80 auch einen residenten COBOL-Compiler; darüberhinaus sind auch residente Compiler für den technisch-wissenschaftlichen Bereich wie FORTRAN und PASCAL geplant.

2.6.9.2. Die Assembler-Sprache

Mit einer Assembler-Sprache hat der Entwickler ein Mittel in der Hand, mit dem er Programme schreiben kann, ohne sich dabei um das Format der Maschinenbefehle (hexadezimale Darstellung und Anzahl der Bytes pro Befehl) und die physikalische Adressierung des Programms zu kümmern braucht. Stattdessen kann die Programmierung symbolisch erfolgen, d.h. jedem Befehl entspricht ein memonischer Code;
Adressen können ebenso in symbolischer Form dargestellt werden, d.h. es brauchen nicht die hexadezimalen Speicheradressen angegeben, sondern es können beliebige Identifikationswörter (z.B. "ZIEL") zur Kennzeichnung einer bestimmten Speicherstelle verwendet werden. Dies beschleunigt die Herstellung des Programms und erleichtert den Test. Darüberhinaus können Marken gesetzt werden ("Labels"), die entsprechend dem vorher über die symbolische Adressierung gesagten dem Programmierer das Auffinden einer bestimmten Speicherstelle ermöglichen, ohne daß die Kenntnis der hexadezimalen, physikalischen Speicheradresse nötig ist. Auch die Operanden in den einzelnen Befehlen können weitgehendst in Symbolischer Darstellung angegeben werden, wodurch Speicherstellen, Register und Konstantenwerte in leicht erfaßbarer Form darzustellen sind.

Die Assembler-Sprache umfaßt außerdem Anweisungen an das Übersetzungsprogramm selbst, die man Pseudo-Befehle nennt. Diese Pseudo-Befehle werden nicht in Maschinenbefehle übersetzt (d.h. es wird von ihnen kein object-code erzeugt),

- eine Marke,
- Operationscode,
- Operanden und
- einen Kommentar.

Dieses Quellenprogramm wird vom Übersetzungs (Assembler-)programm in ein von Mikroprozessor direkt verarbeitbares Objekt-(=Maschinen-)programm übersetzt.

Die Assembler-Sprache des Systems Z80 wurde so aufgebaut, daß sie

- leicht erlernbar ist,
- das Austesten des Programms in einfachster Weise ermöglichen,
- Modularen Programmaufbau ermöglichen,

Dies wurde durch die Einstellung eines Makro-Assemblers erreicht, der mit einer minimalen Anzahl von symbolischen Befehlswörtern auskommt, wobei die Interpretation der Bedeutung der einzelnen Befehle dem Programmierer weitgehendst von der Übersetzungsanlage abgenommen werden wird. So wurde z.B. der Vorgang des Datentransfers grundsätzlich mit einem einzigen symbolischen Befehl realisiert (LD), unabhängig davon, ob diese Datenübertragung zwischen den einzelnen Registern, Registern und Speicherstellen stattfinden oder sogar Konstanten von 8 oder 16Bit Länge in den entsprechenden Registern geladen werden.

Der erste Operand einer Datentransferanweisung ist grundsätzlich die Zieladresse, während der zweite Operand die Sendeadresse dieser Operation ist. Beispiel:

```
LD A, B
```

bewirkt den Transfer des Inhalts des Registers B in das Register A. Entsprechend bewirkt

```
LD C, 03FH
```

das Einschreiben der Konstanten 3FH in das Register C. Darüberhinaus kann man durch Einschließen eines Operanden symbolisieren, daß der Inhalt des Operanden als Adresse interpretiert werden soll, wobei die auf dieser Adresse gefundenen Daten übertragen werden sollen. Beispiel:

```
LD HL, (1200)
```

bewirkt die Übertragung des Inhalts der Speicheradresse 1200 und 1201 in das 16 bit-Registerpaar HL. In ähnlicher Weise wird durch

LD (IX + 6, C)

der Inhalt des Registers C in die Speicheradresse übertragen, die durch den momoentanen Wert des Indexregisters IX plus 6 gegeben ist.

Nach diesem Überblick über die formalen Gegebenheiten der Z80-Assembler-Sprache wollen wir uns nun den Operanden der Z80-Befehle zuwenden.

Operanden beeinflussen die Funktion der Operationscodes und stellen damit den zweiten Teil der Informationen dar, die dem Computer übermittelt werden muß, damit die gewünschte Funktion erfüllt wird:
Während die Operationscodes besagen, was getan werden muß, vermitteln der oder die Operanden, in welcher Weise (also mit welchen Speicherstellen, Registern, Konstanten) die jeweilige Operation durchzuführen ist.

Bestimmte symbolische Namen sind als Schlüsselworte in der Assembler-Sprache für die Kennzeichnung von Operanden reserviert. Diese Schlüsselworte sind:

- die Namen der Register der CPU, nämlich A (für Akkumulator), B, C, D, E, H, L, I R und F (für Flag-Register). Mit diesen reservierten symbolischen Namen werden 8bit Wörter in den zugeordneten Registern adressiert.
- Die Namen der Registerpaare AF, BC, DE, HL, IX, IY und SP. Mit diesen Namen werden jeweils 16bit Werte adressiert, die in Doppelregistern der CPU gespeichert sind. Physikalisch sind die Inhalte dieser Doppelregister die gleichen wie die der gerade zuvor genannten Einzelregister. So bilden beispielsweise die beiden 8bit-Register A und F zusammen das Registerpaar AF, während die beiden 8bit-Register H und L zusammen das 16bit-Register HL bilden. Die nieder- und höherwertigeren Bytes der Doppelregister IX, IY und SP sind nicht wie die der übrigen allgemeinen Register einzeln adressierbar.
- Der Inhalt der Hilfsregisterpaare, die jeweils aus zwei 8bit-Registern bestehen, wird durch ein ' gekennzeichnet. Die Namen der Hilfsregisterpaare sind AF', BC', DE' und HL'. Allerdings kann von diesen Registerpaaren einzig das Registerpaar AF' als ein Operand benützt werden und auch dort nur in dem Austauschbefehl EX AF, AF'.

  Das Arbeiten mit diesem Hilfsregistersatz geschieht durch Übergang vom Hauptregistersatz zum Hilfsregistersatz und umgekehrt über die bereits genannten Austauschbefehle; sobald der Austausch erfolgt ist, werden die Hilfsregister von den normalen Befehlen angesprochen, ohne daß dazu die Angabe eines ' nötig wäre. Der Übergang zurück zu den Hauptregistern erfolgt wiederum durch eine Austauschanweisung.

- Der Zustand der 4 dem Anwender zugänglichen Bedingungsbit ist in der folgenden Tabelle dargestellt. Auch die Namen der Bedingungsbits können in bestimmten Befehl als Operanden vorgegeben werden.

Bedingungsbit	"1"-Bedingung	"0"-Bedingung
Übertrag	C	NC
Null-Bedingungsbit	Z	NZ
Vorzeichen	M(-)	P(+)
Parität	PE (even = Gerade)	PO (odd = Ungerade)

Es sind an dieser Stelle nun noch die formalen Regeln für das Schreiben eines Assembler-Programms zu besprechen (Syntax). Wir haben bereits gesehen, daß das Quellenprogramm, das in ASSEMBLER geschrieben ist, aus Marken, Operationscodes, Operanden und Kommentaren besteht. Anstelle der Operationscodes können die genannten Pseudooperationen treten. Im allgemeinen beginnt ein Quellenprogramm mit einem Pseudobefehl ORG; ist dies nicht der Fall, nimmt der Assembler automatisch eine Anfangsadresse des Maschinenprogramms von "0" an.
Die Z80-Assembler-Sprache umfaßt 74 verschiedene Code-Wörter (wie LD), 25 Schlüsselworte für Operanden (wie A für Akkumulator) und 693 gültige Kombinationen von Op-codes oder Operanden im Rahmen des Z80-Befehlssatzes. Das Format eines Assembler-Befehls ist wie folgt:

Marke	Operationscode	Operand	Kommentar
SCHLEIFE:	LD	HL, ZIEL	;GET VALUE ;ZIEL

In diesem Beispiel stellt die Marke "SCHLEIFE" eine Zuordnung des angegebenen Namens zu dem Befehl LD (Laden eines Wertes) dar und wird dazu verwendet, diesen Befehl von anderen Befehlen her adressieren lassen zu können.
Das Operandenfeld besteht aus einem oder zwei Bestandteilen, die bei einem oder mehreren Kommas oder aber Zwischenräumen getrennt sind.

Das Kommentarfeld wird dazu benützt, um in umgangssprachlicher Form anzugeben, welche Rechneraktivität mit dem jeweiligen Befehl durchgeführt werden soll, um die Transparenz des Programms für den Programmierer und für spätere Änderungen und Korrekturen zu gewährleisten.
An dieser Stelle soll bemerkt werden, daß ausführliche Kommentierung eine der wesentlichsten Kennzeichen eines gut geschriebenen Programms ist. Leider wird häufig die Kommentierung zugunsten der raschen Programmentwicklung vernachlässigt; der geübte Programmierer weiß jedoch, daß die geringe Einsparung an Zeit, die durch das Weglassen von Kommentaren entsteht, durch spätere erheblich höhere "geistige Rüstzeit" und erhöhte Testkosten weit überkompensiert wird.
In der Z80-Assembler-Sprache müssen Kommentare mit einem ";" (Strichpunkt) beginnen.

Marken müssen mit einem Doppelpunkt abgeschlossen werden, falls sie nicht in Spalte eins beginnen. An dieser Stelle soll noch einmal auf die Bedeutung der Pseudobefehle hingewiesen werden.

Pseudoanweisungen sind Anweisungen an das ASSEMBLER-Programm zur Steuerung der Übersetzung des Primärprogramms in das Objektprogramm; folglich gibt es zu Pseudoanweisungen weder einen entsprechenden Z80-Maschinen-Code noch erscheinen sie im Objektcode des Programms oder im Befehlsspeicher des Mikrocomputer-Anwender-Systems. Sie sind jedoch wie ausführbare Anweisungen aufgebaut, können also mit einer Marke versehen und von einem Kommentar gefolgt sein.
Bei der Übersetzung tritt funktionsmäßig der (Software-)-Adreßreferenzzähler an die Stelle des Befehlszählers bei der Ausführung eines Objektprogramms; seine Aufgabe ist die Ermittlung und Zuweisung von Objektcodeadressen für das zu übersetzende Quellprogramm.

Normalerweise beginnt ein Primärprogramm mit einer ORG-Pseudoanweisung; sie legt fest, auf welche Adresse der Beginn des Objektprogramms gelegt wird. Wird sie weggelassen, wird als Anfangsadresse des Programms standardmäßig Speicherstelle Null angenommen.

Die letzte Zeile eines Primärprogramms muß immer eine END-Pseudoanweisung sein; END sagt dem Assembler-Programm, daß die letzte Programmzeile erreicht ist.
Folgende Pseudoanweisungen sind zulässig:

name: ORG adr (="origin")

setzt den Referenzzähler auf den Wert adr. Üblicherweise wird mit dieser Anweisung der physikalische Beginn des Objektprogramms definiert.

name: END

letzte Programmzeile; danach folgende Programmzeilen werden vom ASSEMBLER-Programm ignoriert.

name: EQU adr — (="Equate") weist einer symbolisch angegebenen Marke oder Konstanten "name" den auf den Opcode EQU folgenden Hexadezimalwert "adr" zu. Diese Pseudoanweisung darf in einem Programm für 1 Marke nur einmal vorkommen. Die Angabe von "name" ist optional.

name: DEFL adr — (="Define label") bewirkt das Gleiche wie EQU, darf jedoch in einem Programm mehrmals vorkommen. Dadurch können Marken im Verlauf des Programms unterschiedliche Werte zugewiesen werden.

name: DEFB n — (="Define Byte"): legt den nach dem Opcode angegebenen Hexadezimalwert n auf die nächste verfügbare Speicherstelle. Bei Angabe eines Namens weist sie diesen der durch "name" definierten Speicherstelle zu. Die Angabe von "name" ist optional.

name: DEFW nn	(="Define Word") legt den nach dem Opcode angegebenen Hexadezimalwert als Speicherstelleninhalt fest. Dabei wird das niederwertigere Byte des Operanden auf der Speicherstelle abgelegt, die dem momentanen Wert des Referenzzählers entspricht und das höherwertige Byte auf die nächstfolgende Adresse. Bei der Angabe eines Namens wird er der definierten 16bit-Konstanten zugewiesen. Die Angabe von "name" ist optional.
name: DEFS nn	(="Define Storage") Reserviert die angegebene Anzahl nn ($\leq$ 64k) Bytes ab dem momentanen Wert des Referenzadreßzählers und erhöht den Referenzzähler um diese Anzahl. Die Angabe von "name" ist optional.
name: COND ausdr	Die Pseudoanweisung "COND ausdr" berechnet den Wert von "ausdr". Ist das Ergebnis ungleich Null, wird diese Pseudoanweisung ignoriert.
name: ENDC	Ist das Ergebnis von "ausdr" gleich Null, werden die zwischen den "COND" und einer darauffolgenden "ENDC" Anweisung nicht assembliert.

name: MACRO	$\neq$p0 $\neq$p1 .....$\neq$pn
name: ENDM	Die "MACRO"-Pseudoanweisung weist dem Namen "name" einen Makro mit den aktuellen Werten p0 bis pn zu. Zwischen einer "MACRO" und einem "ENDM" stehende Befehle stellen die Modellanweisung ("Macro-Body") dar. Die Angabe von "name" vor der ENDM-Pseudo-Anweisung ist optional.
name: DEFM 'n'	(="Define Message") definiert auf n Bytesspeicherplätzen den Inhalt dieses Speicherbereichs als ASCH-Äquivalent der auf den Opcode DEFM folgenden Zeichenfolge. Die Länge n dieser Zeichenfolge ist max. 63; sie muß in Hochkommata eingeschlossen sein.

Außerdem sind folgende Formal-Konventionen zu beachten:

Zahlen

können vom Assemblerproramm in den Zahlensystemen binär, oktal, dezimal und hexadezimal verarbeitet und übertragen werden.
Zahlen müssen grundsätzlich mit einer Dezimalziffer (0...9) beginnen und können unmittelbar (d.h. ohne dazwischenliegende Leerstelle) von einem Buchstaben zur Kennzeichnung der Zahlenbasis gefolgt sein, in der die ganze Zahl zu interpretieren ist.

Dabei steht B für "binär '
O oder Q für "oktal"
D für "dezimal" und
H für "hexadezimal".

Ist keine Basiskennung angegeben, wird die betreffende Zahl vom Assemblerprogramm als Dezimalzahl interpretiert. Beispiele gültiger Zahlenangabe:

1011100B
134Q
92 (gleichbedeutend mit 92D)
05CH

Beispiele ungültiger Zahlenangaben

101102B (in Binärzahl nur 0 und 1 erlaubt)
134 Q (zwischen Zahl und Basiskennung kein Leerzeichen erlaubt)
E10FH (Zahl muß mit Ziffer 0....9 beginnen)

Ausdrücke

Der Z80-ASSEMBLER ist zur Übersetzung von arithmetischen und logischen Ausdrücken eingerichtet.
Sie werden grundsätzlich von links nach rechts ausgewertet, soweit die einzelnen Operationen gleiche Bearbeitungsreihenfolge haben.

Die Operationen werden in folgender Reihenfolge ausgeführt:

Begrenzungszeichen (s. oben) wie Komma und Leerzeichen dürfen natürlich nicht in Ausdrücken vorkommen, da sie zu Trennung von Ausdrücken untereinander und zwischen Ausdrücken und übrigen Befehlsteilen dienen.

Operation	Symbol	Priorität
– Positives Vorzeichen festlegen	+	1
– Negatives Vorzeichen festlegen	–	
– Negation	.NOT. oder	
– Resultat ohne Überlauf	.RES.	
– Potenzieren	**	2
– Multiplizieren	*	
– Dividieren	/	
– Modulo	.MOD.	3
– Rechtsschieben	.SHR.	
– Linksschieben	.SHL.	
– Addieren	+	
– Subtrahieren	–	4
– Logisches UND	.AND. oder &	5
– Logisches ODER	.OR. oder	
– Exclusiv ODER	.XOR.	6
– Gleich	.EQ. oder=	
– Größer als	.GT. oder	
– Kleiner als	.LT. oder	7
– Vorzeichenloses "Größer als"	.UGT.	
– Vorzeichenoses "Kleiner als"	.ULT.	

Die fünf Vergleichsoperationen

.EQ.

.GT.

.LT.

.UGT.

.ULT.

liefern als Ergebnis entweder "Wahr" (alle Bits = !) bzw. "Falsch" (alle Bits = 0): dabei arbeiten .GT. und .LT. mit vorzeichenbehafteten .UGT. und.ULT. mit vorzeichenlosen Zahlen.

Der Ergebnisoperator

.RES.

bewirkt, daß die Überlaufmeldung unterdrückt wird.

Zum Beispiel würde der Befehl

LD BC, 7FFFH + 1

eine Fehlermeldung bei der Übersetzung veranlassen, während bei

LD BC, .RES. 7FFFH + 1

keine Fehlermeldung erfolgt.

Der Modulooperator

.MOD.

ist durch folgende Entsprechung definiert:

X. MOD. Y = X – Y x (X/Y)

wobei X/Y als eine Division mit ganzzahligen Ergebnis aufzufassen ist.

Die Schiebeoperatoren

.SHR.

.SHL.

schieben ihr erstes Argument nach rechts respektive nach links, und zwar um soviele Stellen, wie das zweite Argument angibt. Die freien Stellen werden dabei mit Nullen aufgefüllt. Beachten Sie bitte, daß auch die Verwendung von Klammern zur Festlegung der Bearbeitungsreihenfolge möglich ist, daß aber das Setzen von Klammern um einen ganzen Ausdruck den Inhalt einer Speicheradresse bezeichnet.
Sämtliche Operationen werden mit ganzen Zahlen ("integers") Assembler-Programm der Momentanwert des Befehlszählers (=Adresse des nächsten auszuführenden Befehls) von der Sprungzieladresse zu subtrahiert.

Beispiel: JR NC, LOOP

führt einen relativen Sprung zu der Speicheradresse durch, die durch die Marke "LOOP" gekennzeichnet ist.

Bei Überschreitung des für Ausdrücke zugelassenen Speicherbereichs (für relative Sprünge ist dies 128 Bytes) wird vom ASSEMBLER eine Fehlermeldung ausgegeben.

Eine generelle Eigenschaft des Assemblerprogramms ist die Möglichkeit, durch Angabe des Dollarzeichens ($) Bezug auf den momentanen Wert des Befehlszählers zu nehmen. Dadurch kann auf einfache Weise eine Distanz im Bereich (– 126, + 129) zur relativen Festlegung eines Sprungzeichens angegeben werden. Zum Beispiel bewirkt

JR C, + 5

einen Sprung auf eine Speicherstelle, die 5 Byte unter dieser Anweisung liegt.

Marken

sind Symbole, die für 16 bit-Speicher-Adressen oder bis zu 16 bit langen Datenwörtern stehen. Wird eine für ein >8 bit langes Wort definierte Marke für eine 8 bit

Konstante benützt, gibt der ASSEMBLER die Fehlermeldung "OUT OF RANGE" aus.
Marken dürfen beliebig lang sein, wobei allerdings nur die ersten 6 Zeichen relevant sind und das erste Zeichen ein Buchstabe sein muß; als weitere Zeichen sind Buchstaben, Ziffern, Fragezeichen und das Unterstreichungszeichen zulässig. Alle anderen Zeichen innerhalb einer Marke veranlassen eine Fehlermeldung des Assembler-Programms.
Marken dürfen bis zu 6 Zeichen lang sein, wobei das erste Zeichen ein Buchstabe sein muß; die Sonderzeichen = > + – * / () & S . , ' ; oder: dürfen nicht vorkommen.
Marken können in jeder beliebigen Spalte beginnen, soweit sie mit einem Doppelpunkt abgeschlossen sind. Beginnen sie in der ersten Spalte, ist der Doppelpunkt nicht nötig.
Die zugehörige Symboltabelle steht normalerweise im Arbeitsspeicher (RAM). Bei Überlauf des hierfür verfügbaren Bereichs wird automatisch auf die Systemdiskette ausgelagert., so daß die Anzahl der verarbeitbaren Marken unbeschränkt ist.

Groß/Kleinschreibung

Das Assemblerprogramm kann Primärprogrammtexte verarbeiten, in denen groß- und klein geschriebene Zeichen vorkommen.
Dabei gelten folgende Regeln:

- Alle Opcodes und Schlüsselwörter (wie Registernamen oder Bedingungscodes) müssen entweder durchgehend groß oder klein geschrieben werden.
- Marken dürfen jede beliebige Mischung aus groß- und klein geschriebenen Zeilen enthalten; unterschiedliche Schreibweise bewirkt aber auch unterschiedliche Interpretation der Marken.

So werden beispielsweise die Notationen

LABEL1

LabEl1

lAbEl1

als verschiedene Marken interpretiert.

Dadurch ist ohne Redefinition die Verwendung von symbolischen Opcodes als Makro- oder Unterprogramme möglich (Beispiel Push, LdiR).

- Alle Assemblerkommandos (z.B. *LIST oder *INCLUDE) können in einer beliebigen Folge von groß- und kleingeschriebenen Zeichen gegeben werden.
- Das gleiche gilt für die Arithmetikoperatoren (wie .AND., .EQ. u.s.w.) und für sämtliche Zahlen (Beispiel. 0fffH, 10ah u.s.w.).

2.6.9.3. Unterprogrammtechnik

Programmteile, die immer wieder in unveränderter Form verwendet werden, erstellt man zweckmäßigerweise nur ein einziges Mal und fügt sie bei Bedarf in das Anwender-Arbeitsprogramm ein. Prinzipiell stehen zum Einfügen solcher Programmstücke zwei verschiedene Methoden zur Verfügung:

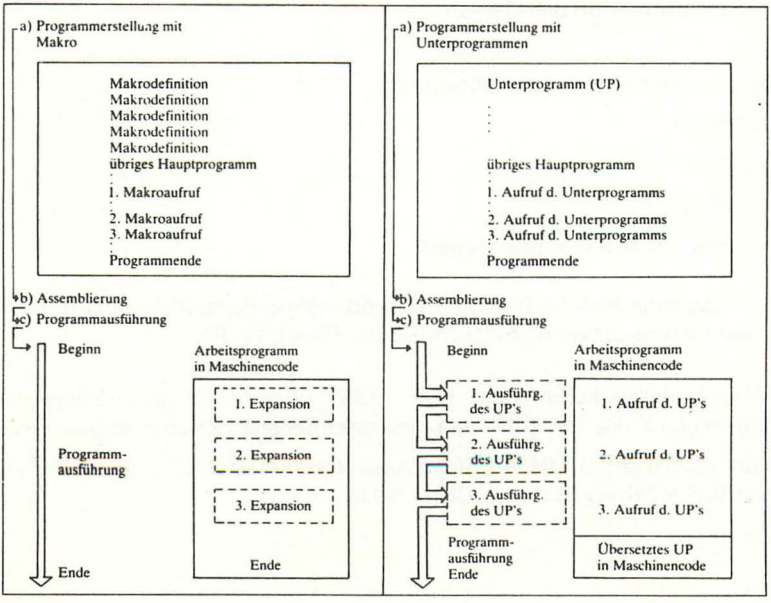
- Unterprogramme (Subroutinen) und
- Makros.

Sowohl Makro als auch Unterprogramm sind Programmabschnitte, die über einen Aufruf im Arbeitsprogramm aktiviert werden. Das untenstehende Bild 23 zeigt:

- die verschiedene Behandlung eines Makros und eines Unterprogramms durch das Assemblerprogramm und
- das unterschiedliche Verhalten des Anwenderprogramms in dem ein Makro bzw. ein Unterprogramm aufgerufen wurde.

Makros werden während der Übersetzung bei jedem Makroaufruf in ihrer ganzen Länge in das Anwenderprogramm eingefügt. Dadurch wird das ablauffähige Anwenderprogramm natürlich um so viel länger, je öfter darin Makroaufrufe vorkommen.

Bild 23: Vergleich zwischen Makros und Unterprogrammen



Unterprogramme
treten im Anwenderprogrammlisting nur als ein einziger separater Programmbestandteil auf, der bei jedem Aufruf ausgeführt wird, ohne daß seine Befehle bei jedem Aufruf in das Hauptprogramm eingefügt worden wären.
(d.h., daß relativ wenig Speicherplatz aufgewandt werden muß, sobald der Programmteil mehr als einmal aufgerufen wird), während Makros **dynamisch günstiger** sind (d.h. Makroaufrufe ermöglichen die Durchführung gleicher Aufgaben in relativ kürzerer Zeit, da sie weniger Zeit beanspruchen als Unterprogrammaufrufe).
Man setzt daher Unterprogramme **dann** ein, wenn Routinen sehr häufig verwendet werden oder minimaler Programmspeicherbedarf ausschlaggebend ist, während Makros bei maximaler Verarbeitungsgeschwindigkeit zu verwenden sind.

Folgende **Begriffe** werden in **der Makro-Programmiertechnik** verwendet:

- Die **Makrodefinition** spezifiert die Befehlsfolge die durch den Makronamen repräsentiert werden soll. Zu beachten ist, daß ein Makro innerhalb eines Programms nur einmal definiert werden darf. Der
- **Makroaufruf** ist der Punkt in einem Programm, an dem der Makro eingefügt werden soll. Ein Makro kann beliebig oft aufgerufen werden, indem man den Makronamen einfach wie ein Op-Code im Anwenderprogramm verwendet.
- **Makroexpansion** ist der Programmabschnitt, den der ASSEMBLER anstelle jedes Makroaufrufs im Arbeitsprogramm generiert.
- **Makroparameter** bezeichnen im Gegensatz zu den Platzhaltern ("Dummyparameters") die Parameter selbst, die in der Operandenliste der Makrodefinition angegeben werden. Beim Makroaufruf werden ihnen durch die Liste im Operandenfeld aktuelle Werte zugewiesen, die der Assembler jeweils für den Parameter an jeder Stelle einsetzt.
- **Modellanweisung** ("Makrobody") bezeichnet die Gesamtheit der Anweisungen, die in der Makrodefinition zwischen dem Pseudobefehl "MAKRO" und "ENDM" stehen.
- **Platzhalter** ("Dummy parameters") bezeichnen die Plätze innerhalb der Modellanweisung, an denen bei der Assemblierung an der jeweiligen Makroaufrufstelle die aktuellen Werte dem Makroparometer einzufügen sind.

2.7. Die Z80 A-CPU

Am Beispiel eines der leistungsfähigsten - und dabei typischsten Mikroprozessoren- der Z80A-CPU, wollen wir hier unsere Mikrocomputergrundlagen etwas mehr konkretisieren.

Dieser Prozessor baut auf den "Urahnen" der Mikrocomputertechnik, nämlich dem System 8008 und dem System 8080 auf und ist zu dem hardwaremäßig etwas weiterentwickelten System 8085 kompatibel, wobei er dem Anwender softwaremäßig bzw. archtekturell noch weitere wesentliche Erleichterungen (und damit höhere Wirtschaftlichkeit bezüglich Entwicklungsaufwand und Programmspeicherbedarf!) bietet.
Der Prozessor ist außerdem so gestaltet, daß er den lückenlosen Anschluß (Kompatibilität) auf der Hardware- und Software-Seite zu den Einchipmikrocomputer- (ZILOG Z8) und 16bit-Mikroprozessorlösungen seiner Herstellerfirma garantiert.
Seine hervorstechendsten Eigenschaften sollen hier kurz umrissen werden:

- 1 Chip-Mikroprozessor in n-Kanal-Silicon-Gate-Technologie.
- Befehlssatz umfaßt 158 Befehle, darunter sämtliche 78 8080-Instruktionen (voll Software-kompatibel!).
  Über den 8080-Befehlssatz hinaus verfügt der Z80 über umfassende 16-, 8-, 4- und Einzel-Bit-Instruktionen und zusätzliche Adressierweisen (indizierte, relative und Bit-Adressierung).
- 17 interne Register.
- 3 Interrupt-Behandlungsarten und ein zusätzlicher, schneller, nichtmaskierbarer Interrupt.
- Direkter Anschluß von dynamischen oder statischen Standardspeicherchips ohne zusätzlichen Bauteileaufwand.
- Eingebaute Refresh-Controller für dynamische Speicherbausteine
- Befehlsausführungsdauer: 1 Mikrosekunde minimal.
- Stromversorgung über eine einzige 5-V-Versorgungsspannung.
- 5-V-Einphasen-Takt.
- Alle Anschlüsse TTL-kompatibel.

Die Z80A-CPU (Ionenimplantierter n-Kanal-Silicon-Gate-Schaltkreis) ist ein 8bit-Mikroprozessor mit bisher unerreichter Leistungsfähigkeit.
Der gegenüber bisher verfügbaren Mikroprozessoren höhere Informations-Durchsatz ("throughput") und die effizientere Programmspeicherausnutzung erlauben die Realisierung von Mikroprozessoren-Systemen mit höheren Anforderungen als bei Systemen mit Mikroprozessoren der 2. Generation.
Darüberhinaus vereinfacht Z80 durch die Notwendigkeit nur einer einzigen Speisespannung und der direkten Anschlußmöglichkeit preiswerter Standardspeicherbausteine den nötigen Schaltungsaufwand.

Bild 4 zeigt ein Blockschaltbild der CPU. In Bild 5 ist die Registerorganisation (= insgesamt 208 Bit dem Anwender zugänglicher Schreib/Lesespeicher) darstellt.
Hiervon sind zwei gleichaufgebaute Blöcke mit je 6 allgemeinen Registern, die jeweils wahlweise 8- oder 16-Bit-Operationen erlauben.
Hinzu kommen zwei Akkumulatoren und Status ("Flag"-) Register.
Die Operationen der CPU laufen in einem der beiden Register/Akku-Blöcke ab; der Zugriff zum zweiten Register/Akku-Block erfolgt durch Übergangs-("Exchange"-) Anweisungen.

Diese alternative Arbeitsweise ermöglicht wechselweises Arbeiten in Haupt- und Hintergrundprogramm ohne Auslagern von Registerinhalten in den Arbeitsspeicher und dadurch besonders schnelle und effiziente Interruptbehandlung.
Der 16-Bit-Stack-Pointer der CPU dient zur Bearbeitung von Mehrebenen ("Multilevel-")Interrupts, praktisch unbegrenzte Unterprogramm-Verschachtelung ("Subroutine Nesting") und Zwischenspeicherung von Datenblöcken.
Zwei 16-Bit-Indexregister erlauben die Bearbeitung von Tabellen und relokativen (d.h. im Adreßraum des Arbeitsspeichers verschiebbaren) Informationen.
Ein eigenes Refresh-Register wurde für ein direktes, übersichtliches Arbeiten mit externen dynamischen Speichern ohne Software-Aufwand implementiert.
Das Unterbrechungsregister ("I-Register") schließlich liefert zur Realisierung einer besonders leistungsfähigen Art der Interrupt-Behandlung die höherwertigen 8 bits eines Zeigers, der über eine Tabelle den Sprung in Interrupt-Behandlungsprogramme ermöglicht; die niederwertigen 8 Bit der Anfangsadressen werden in üblicher Weise von den anfordernden Schaltung geliefert.

Zu den genannten architektonischen Eigenschaften kommen folgende Softwarebesonderheiten:

- Befehle zur Behandlung von 4 bit, 8 bit und 16 bit Datenwörter
- Befehle zum Register rotieren **und** schieben

Als einziger Prozessor ist die Z80-CPU in der Lage, sowohl Einzelbits als auch ganze Dateien mit einem einzigen Befehl zu bearbeiten (Blocktransfers mit einer Blockgröße bis zu 64 kByte bei einer Übertragungsrate von 5,3 μsec/Byte, Blocksuchbefehl, Block-Ein/Ausgaberate von 200 kByte/sec., Setzen, Testen oder Rücksetzen irgendeines einzelnen Bits in einen der CPU Register oder einer Speicherstelle).

Durch all diese Eigenschaften werden Programmentwicklungs- und Testkosten eingespart, was besonders bei in kleinen Stückzahlen gefertigten Geräten ausschlaggebend ist, gleichzeitig wird die Anzahl der System benötigten Programmspeicherbausteine reduziert (wichtig für Systeme in großen Stückzahlen) und die Verarbeitungsgeschwindigkeit in hohem Maß gesteigert.

Da, wie schon erwähnt, u.a. die Interrupt-Bearbeitungsfähigkeit des Systems besonders hochqualifiziert ist, eröffnet dieser Prozessor den preisgünstigen weit verbreiteten NMOS-Mikrocomputertechnik Anwendungsmöglichkeiten, die bisher den teuren Prozeßrechnern oder den aufwendig zu handhabenden bipolaren Bit-Slice-Mikroprozessoren vorbehalten waren. Doch nun zurück zu den technischen Eigenschaften der Zentraleinheit dieses Systems.

Wir haben bereits in den vorigen Abschnitten die große Bedeutung der Anschlußweise der Halbleiterbausteine kennengelernt. Dies gilt in besonderem Maße auch für die Mikrocomputerbausteine. Die Funktion der Anschlüsse der Z80-CPU ist wie folgt:

Signal	Funktion	Kommentar
A_0-A_{15}	Adreßbus	Tri-State Ausgänge, High-aktiv. Liefern die Adressen für den Speicher (bis zu 64 kByte) und die Ein/Ausgabebausteine.
D_0-D_7	Datenbus	Tri-State Ein/Ausgänge (Bidirektional), High-aktiv. Über den Datenbus erfolgt der Datenaustausch zwischen CPU und Speicher bzw. CPU und E/A-Bausteinen.
$\overline{M1}$	Maschinenzyklus Eins	Ausgang, Low-aktiv. M 1 = aktiv bedeutet, daß der momentane Maschinenzyklus der Operationscode-Lesezyklus (Fetch-Zyklus) der momentan auszuführenden Anweisung ist.
$\overline{MREQ}$	Speicheranforderung (= „Memory-Request")	Tri-State Ausgang, Low-aktiv. MREQ = aktiv bedeutet, daß auf dem Adreßbus die Adresse für einen Speicherzugriff (Lesen oder Schreiben) ansteht.
$\overline{IOREQ}$	Ein/Ausgabe-Anforderung (= „Input/Output-Request")	Tri-State-Ausgang, Low-aktiv. IOREQ = aktiv bedeutet, daß an den niederwertigen 8 bits des Adreßbus eine Adresse zur I/O-Portauswahl (Eingabe oder Ausgabe) ansteht. Ein IOREQ-Signal wird auch dann erzeugt, wenn eine Interruptanforderung akzeptiert wurde; in diesem Fall kann dann der zugehörige Interrupt-Vektor auf den Datenbus gelegt werden.
$\overline{RD}$	Lesen („Read")	Tri-State-Ausgang, Low-aktiv. RD = aktiv bedeutet, daß die CPU Daten vom Speicher oder von einem I/O-Port lesen soll. Der angesprochene Speicher oder I/O-Baustein interpretiert das Signal als Aufforderung, Daten auf den Datenbus zu legen.
$\overline{WR}$	Schreiben („Write")	Tri-State-Ausgang, Low-aktiv. WR = aktiv bedeutet, daß die CPU Daten für den Speicher oder einen I/O-Baustein auf dem Datenbus bereithält.
$\overline{RFSH}$	Refresh	Ausgang, Low-aktiv. RFSH = aktiv bedeutet, daß die niederwertigen 7 bit des Adreßbus eine Refreshadresse für dynamische Speicher führen und das laufende MREQ-Signal zur Einleitung eines Refresh-Zyklus für alle angeschlossenen dynamischen Speicher zu benutzen ist.
$\overline{HALT}$	Halt-Zustand	Ausgang, Low-aktiv. HALT = aktiv bedeutet, daß die CPU einen (Software-)HALT-Befehl ausgeführt hat und zur weiteren Abarbeitung des Programms auf ein Interrupt-Signal wartet (einen nicht-maskierbaren Interrupt oder aber einen freigegebenen maskierbaren Interrupt). Im Halt-Zustand führt die CPU zur Sicherstellung des Refresh-Vorgangs Leerbefehle (NOP's) aus.
$\overline{WAIT}$	Warte-Signal („Wait")	Eingang, Low-aktiv. Low-Signal am WAIT-Eingang zeigt der CPU, daß die angesprochenen Speicher- oder I/O-Bausteine noch nicht zur Datenübertragung bereit sind. Die CPU beginnt mit Wait-Zyklen, solange der Eingang aktiviert wird.
$\overline{INT}$	Interrupt-Eingang („Interrupt Request")	Eingang, Low-aktiv. Das Interrupt-Anforderungssignal wird von einer peripheren Schaltung erzeugt. die Anforderung wird nach Abarbeitung des in Ausführung befindlichen Befehls berücksichtigt, soweit das interne Software-gesteuerte Interrupt-Freigabe-Flip Flop gesetzt und das BUSRQ-Signal nicht aktiv ist.
$\overline{NMI}$	Nicht maskierbarer Interrupt („nonmaskable interrupt")	Eingang, Low-aktiv. Eine Interrupt-Anforderung auf diesem Eingang hat höhere Priorität als Interruptanforderungen auf dem Eingang INT für maskierbare Interrupts und wird durch das interne Interrupt-Freigabe-Flip Flop nicht beeinflußt. Gelangt ein Low-Signal an den NMI-Eingang, so wird entsprechend einer RESTART-Instruktion die Programmbehandlung bei Speicheradresse 0066H fortgesetzt.
$\overline{RESET}$	Rückstellen („Reset")	Eingang, Low-aktiv. Bewirkt Rücksetzen (= 0) von Interrupt-Freigabe-Flip-Flop, Befehlszähler, Register I und R und bringt die CPU in die 8080-Interrupt-Betriebsart. Während des Rückstellungsvorgangs befinden sich Daten- und Adreßbus im hochohmigen, sämtliche übrigen Ausgänge im inaktiven Zustand.
$\overline{BUSRQ}$	Bus-Anforderung („Bus Request")	Eingang, Low-aktiv. Bringt Adreß-, Daten- und Steuerbus-Signal in den hochohmigen Zustand, sodaß externe Schaltungen diese Leitungen benutzen können.
$\overline{BUSAK}$	Bus-Anforderungsbestätigung („Bus Acknowledgement")	Ausgang, Low-aktiv. Bestätigt, daß Adreß-, Daten- und Steuerbus in den hochohmigen Zustand gebracht wurden.

In der folgenden Zeichnung ist das Zeitverhalten der Z80-CPU bzw.
gezeigt.
Es sind folgende Arten von Verarbeitungszyklen zu unterscheiden:

- **Befehlslesezyklus ("Instruction Op-Code-Fetch")**

Der Inhalt des Befehlszählers wird bei Beginn des Zyklus auf den Adreßbus gebracht. 1/2 Taktimpuls später wird $\overline{\text{MREQ}}$ aktiv, so daß die fallende Flanke des $\overline{\text{MREQ}}$ direkt als Freigabe-Signal ("Chip Enable") für dynamische Speicherbausteine benutzt werden kann Während $\overline{\text{RD}}$ = aktiv müssen die Daten aus dem Speicher

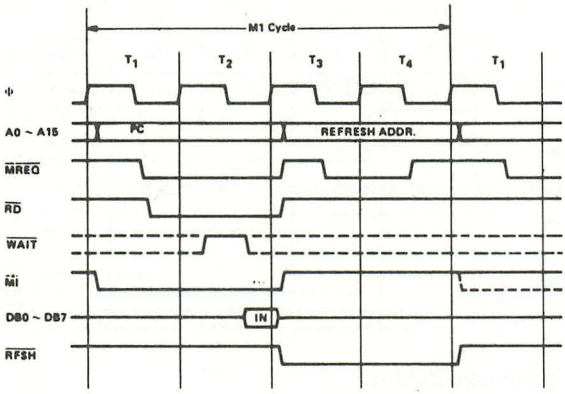
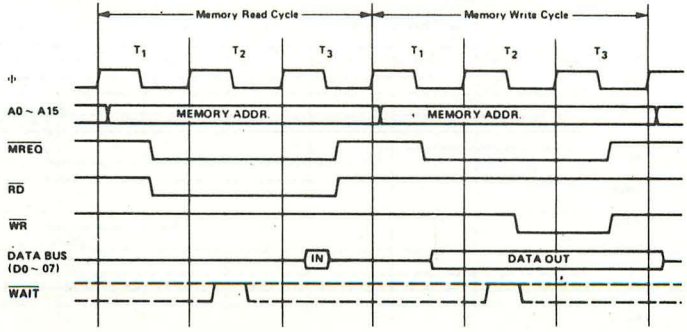


Bild 24: CPU-Befehlslesezyklus

auf den Datenbus gebracht werden; die CPU holt diese Daten mit der steigenden Flanke des Taktimpulses T3 ab. Taktimpulse T3 und T4 eines Lesezyklus' werden benutzt, um dynamischen Speichern während des Befehlsdekodier- und Ausführungsvorgangs in der CPU das Refresh-Signal zu liefern.
$\overline{\text{RFSH}}$ gibt diese Refresh-Signale frei.

- **Speicherzugriffszyklen ("Memory-Read-or-Write")**

Bei Lese- oder Schreibzyklen verhalten sich die Signale $\overline{\text{MREQ}}$ und $\overline{\text{RD}}$ genauso wie beim Befehlszyklus. Beim Schreibzyklus wird $\overline{\text{MREQ}}$ aktiv, sobald die Infor-



mation auf dem Adreßbus stabil ist, so daß dieses Signal direkt als Bausteinauswahlsignal ("Chip-Enable") für dynamische Speicher verwendet werden kann.
$\overline{WR}$ ist aktiv, sobald die Informationen auf dem Datenbus stabil sind, so daß er für praktisch alle Halbleiterspeicher als R/W-Signal zu verwenden ist.

● **Ein/Ausgabezyklus ("Input or Output Cycles")**

Während Ein/Ausgabe Operationen wird automatisch ein Wartezyklus (T_W*) eingefügt, um den I/O Port genügend Zeit zur Adreßdekodierung und eventueller Aktivierung des WAIT-Signals zu geben.

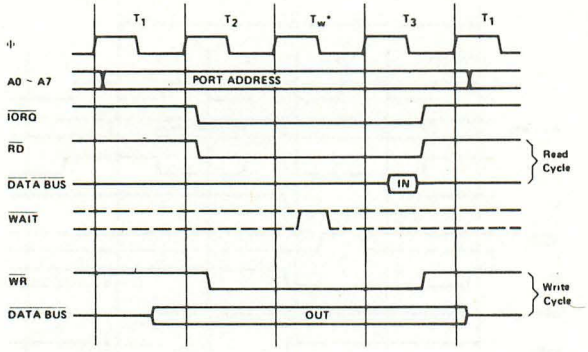


Bild 26: CPU-Ein/Ausgabe-Zyklus

● **Unterbrechungsbehandlungs-Zyklen ("Interrupt Request/Acknowledge Cycle")**

Der Interrupt-Eingang wird von der CPU bei der steigenden Flanke des letzten Taktimpulses jeder ausgeführten Anweisung abgefragt. Im Interrupt-Fall wird dann ein besonderer $\overline{M1}$-Zyklus erzeugt, bei dem $\overline{IORQ}$ statt $\overline{MREQ}$ aktiv wird als Signal für die unterbrechende Schaltung, daß nun ein 8-Bit-Vektor auf den Datenbus zu legen ist. Zwei Wartezyklen Tw* werden hierbei automatisch eingefügt, um der Prioritätsschaltung genügend Zeit zum Durchschalten der Prioritätssignale zu geben.

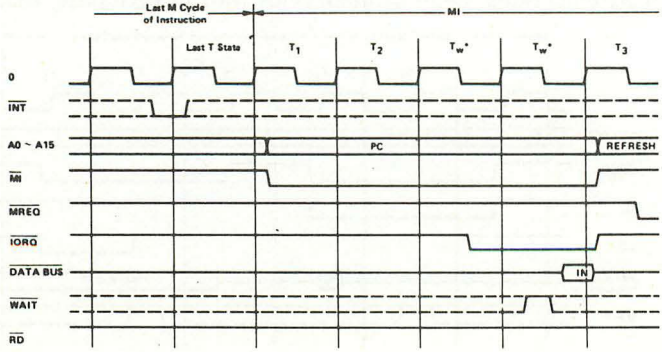


Bild 27: CPU-Interrupt-Zyklus

● Z80A-CPU und Z80-CPU-Befehlssatz

Im folgenden wird lediglich ein großer Überblick über den Z80-Befehlssatz gegeben. Einzelheiten finden Sie im Z80-Manual. Zu unterscheiden sind folgende Befehlsgruppen:

- o 8 Bit-Ladebefehle (="8 bit loads")
- o 16 Bit-Ladebefehle (="16 bit loads")
- o Austauschbefehle (="Exchanges")
- o Blocktransfers im Speicher ("Memory Block Moves")
- o Blocksuchbefehle ("Memory Block Searches")
- o 8 Bit arthmetische und logische Befehle (="8 bit arithmetic and logic")
- o 16 Bit arithmetische Befehle (="16 bit arithmetic)
- o Allgemeine Akkumulator- und Status-Anweisungen (="General purpose Accu and Flag Operations")
- o Akku-Rotieren und -Schieben (="Rotate and Shift")
- o Bit Setzen, Rücksetzen und Testen (="Bit Set, Reset and Test")
- o Ein/Ausgabe (="Input and Output")
- o Sprünge (="Jumps")
- o Unterprogrammaufrufe ("Calis")
- o Restarts ("Restarts")
- o Rücksprünge (="Returns")
- o Sonstige Befehle ("Miscellancous Group")

Zeichenerklärung;

b	bezeichnet eine Bit-Position in einem Register; oder einer Speicherstelle;
cc	Statusbedingungscode ("Flag condition");
	Erlaubte Bedingungen:;
	NZ: ungleich Null (="Nonzero");
	Z: gleich Null (="Zero");
	NC: Kein Übertrag (="Non carry");

C: Übertrag (="carry");

PO: Ungerade oder kein Überlauf ("Parity Odd")

PE: Gerade oder Überlauf ("Parity Even")

P: Positiv

N: Negativ

d	Zielregister (8 bit)
dd	16 bit-Zielregister oder Zieladresse im Speicher
e	8 bit vorzeichenbehaftetes Zweierkomplement der Distanz bei relativen Sprüngen oder indizierter Adressierung
L	bezeichnet die 8 speziellen Zieladressen in Seite O (dezimal O, 8, 16, 32, 40, 48 und 56)
n	8 bit-Binärzahl
nn	16 bit-Binärzahl
r	allgemeines 8 bit-Register (A, B, C, D, E, H oder L)
s	8 bit Senderregister oder Speicherstelle
sb	ein Bit in einem bestimmten 8 bit-Register oder Speicherstelle
ss	16 bit-Senderegister oder Speicherstelle
Index 'L"	Niederwertige (="Low order") 8 bits eines 16 bit-Registers
Index "H"	Höherwertige ("High order") 8 bits eines 16-bit-Registers
( )	"Inhalt von ...."

Zeichen zwischen den Klammern stellen einen Zeiger auf eine Speicherstelle oder ein I/O Port dar.

8 bit Register sind: A, B, C, D, E, H, L, I und R.

16 bit Registerpaare: AF, BC, DE und HL.

16 bit Register: SP, PC, IX und IY.

	Mnemo	Durchgeführte Operation	Bemerkungen
8 bit-Ladebefehle	LD r, s	r ← s	s ≡ r, n, (HL), (IX+e), (IY+e)
	LD d, r	d ← r	d ≡ (HL), r (IX+e), (IY+e)
	LD d, n	d ← n	d ≡ (HL), (IX+e), (IY+e)
	LD A, s	A ← s	s ≡ (BC), (DE), (nn), I, R
	LD d, A	d ← A	d ≡ (BC), (DE), (nn), I, R
16 bit-Ladebefehle	LD dd, nn	dd ← nn	dd ≡ BC, DE, HL, SP, IX, IY
	LD dd, (nn)	dd ← (nn)	dd ≡ BC, DE, HL, SP, IX, IY
	LD (nn), ss	(nn) ← ss	ss ≡ BC, DE, HL, SP, IX, IY
	LD SP, ss	SP ← ss	ss = HL, IX, IY
	PUSH ss	$(SP-1) \leftarrow ss_H$; $(SP-2) \leftarrow ss_L$	ss = BC, DE, HL, AF, IX, IY
	POP dd	$dd_L \leftarrow (SP)$; $dd_H \leftarrow (SP+1)$	dd = BC, DE, HL, AF, IX, IY
Registeraustausch	EX DE, HL	DE ↔ HL	
	EX AF, AF'	AF ↔ AF'	
	EXX	$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \leftrightarrow \begin{pmatrix} BC' \\ DE' \\ HL' \end{pmatrix}$	
	EX (SP), ss	$(SP) \leftrightarrow ss_L$, $(SP+1) \leftrightarrow ss_H$	ss ≡ HL, IX, IY
Block-Suchbefehle	LDI	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1	
	LDIR	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1 Repeat until BC = 0	
	LDD	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1	
	LDDR	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1 Repeat until BC = 0	
	CPI	A-(HL), HL ← HL+1 BC ← BC-1	
	CPIR	A-(HL), HL ← HL+1 BC ← BC-1, Repeat until BC = 0 or A = (HL)	A-(HL) sets the flags only. A is not affected
	CPD	A-(HL), HL ← HL-1 BC ← BC-1	
	CPDR	A-(HL), HL ← HL-1 BC ← BC-1, Repeat until BC= 0 or A = (HL)	
8 bit-arithmetische und logische Operationen	ADD s	A ← A + s	
	ADC s	A ← A + s + CY	CY is the carry flag
	SUB s	A ← A - s	
	SBC s	A ← A - s - CY	s ≡ r, n, (HL) (IX+e), (IY+e)
	AND s	A ← A ∧ s	
	OR s	A ← A ∨ s	
	XOR s	A ← A ⊕ s	

16 bit-arithmetische Operationen

Mnemo	Durchgeführte Operation	Bemerkungen
CP s	A - s	s = r, n (HL) (IX+e), (IY+e)
INC d	d ← d + 1	d = r, (HL) (IX+e), (IY+e)
DEC d	d ← d - 1	
ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE HL, SP
ADC HL, ss	HL ← HL + ss + CY	
SBC HL, ss	HL ← HL - ss - CY	
ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP
ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
DEC dd	dd ← dd - 1	dd ≡ BC, DE, HL, SP, IX, IY

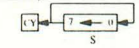
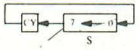
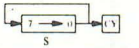
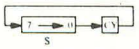
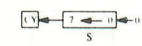
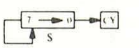
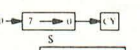
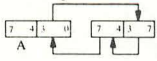
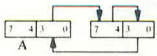
BCD-, Akku- und Flag-Operationen

Mnemo	Durchgeführte Operation	Bemerkungen
DAA	Converts A contents into packed BCD following add or subtract.	Operands must be in packed BCD format
CPL	$A \leftarrow \overline{A}$	
NEG	A ← 00 – A	
CCF	$CY \leftarrow \overline{CY}$	
SCF	CY ← 1	

Verschiedene

Mnemo	Durchgeführte Operation	Bemerkungen
NOP	No operation	
HALT	Halt CPU	
DI	Disable Interrupts	
EI	Enable Interrupts	
IM 0	Set interrupt mode 0	8080A mode
IM 1	Set interrupt mode 1	Call to 0038_H
IM 2	Set interrupt mode 2	Indirect Call

Rotate- und Schiebeoperationen

Mnemo	Durchgeführte Operation	Bemerkungen
RLC s		s ≡ r, (HL) (IX+e), (IY+e)
RL s		
RRC s		
RR s		
SLA s		
SRA s		
SRL s		
RLD		
RRD		

Bit Setzen, Rücksetzen, Testen

Mnemo	Durchgeführte Operation	Bemerkungen
BIT b, s	$Z \leftarrow \overline{s_b}$	Z is zero flag
SET b, s	$s_b \leftarrow 1$	s ≡ r, (HL) (IX+e), (IY+e)
RES b, s	$s_b \leftarrow 0$	

Ein/Ausgabe

Mnemo	Durchgeführte Operation	Bemerkungen
IN A, (n)	A ← (n)	
IN r, (C)	r ← (C)	Set flags
INI	(HL) ← (C), HL ← HL + 1 B ← B - 1	
INIR	(HL) ← (C), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
IND	(HL) ← (C), HL ← HL - 1 B ← B - 1	
INDR	(HL) ← (C), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
OUT(n), A	(n) ← A	
OUT(C), r	(C) ← r	
OUTI	(C) ← (HL), HL ← HL + 1 B ← B - 1	
OTIR	(C) ← (HL), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
OUTD	(C) ← (HL), HL ← HL - 1 B ← B - 1	
OTDR	(C) ← (HL), HL ← HL - 1 B ← B - 1 Repeat until B = 0	

Sprungbefehle

Mnemo	Durchgeführte Operation	Bemerkungen
JP nn	PC ← nn	
JP cc, nn	If condition cc is true PC ← nn, else continue	cc ≡ NZ, Z, NC, C, PO, PE, P, M
JR e	PC ← PC + e	
JR kk, e	If condition kk is true PC ← PC + e, else continue	kk ≡ NZ, Z, NC, C
JP (ss)	PC ← ss	ss = HL, IX, IY
DJNZ e	B ← B - 1, if B = 0 continue, else PC ← PC + e	

Unterprogramm-aufruf

Mnemo	Durchgeführte Operation	Bemerkungen
CALL nn	(SP-1) ← PC_H (SP-2) ← PC_L, PC ← nn	
CALL cc, nn	If condition cc is false continue, else same as CALL nn	cc ≡ NZ, Z, NC, C, PO, PE, P, M

Restarts

Mnemo	Durchgeführte Operation	Bemerkungen
RST L	(SP-1) ← PC_H (SP-2) ← PC_L, PC_H ← 0 PC_L ← L	

Rücksprünge

Mnemo	Durchgeführte Operation	Bemerkungen
RET	PC_L ← (SP), PC_H ← (SP+1)	
RET cc	If condition cc is false continue, else same as RET	cc ≡ NZ, Z, NC, C, PO, PE, P, M
RETI	Return from interrupt, same as RET	
RETN	Return from non-maskable interrupt	

2.8. Speicher für Programm und Daten

2.8.1. Allgemeines

Für alle Speicherbausteine hat die Frage der **Speicherorganisation** zentrale Bedeutung. Damit ist gemeint, wieviele Bits durch Anlegen einer bestimmten Adresse an den Adreßvorgängen des Bausteins gleichzeitig adressiert und damit ausgelesen respektive eingeschrieben werden können.

Beispiele:

1k x 4 bedeutet, daß der Baustein 1024 (=1k) Worte (=Speicherzellen) a 4 bit enthält, die gleichzeitig über 4 parallele Datenleitungen bei Vorgabe einer 10 bit-Adresse ($1024=2^{10}$) angesprochen sind

16k x 1 bedeutet, daß der Baustein über 16384 (=16k) Worte (=Speicherzellen) a 1 bit verfügt. Jede dieser 1 Bit-Speicherzellen wird durch die Vorgabe einer 14 bit-Adresse ($16384=2^{14}$) angesprochen.

Um einen Speicher für einen n-bit-Computer zu realisieren, muß man dann mindestens soviele Bausteine verwenden, daß n-bit gleichzeitig (parallel) zur Verfügung stehen.

Beispiel: Speicher für einen 8 bit-Mikrocomputer

Realisierung mit 1k x 4 Bausteinen: Mindestens 2 Bausteine nötig. Die Speicherkapazität beträgt in diesem Fall 1kByte (= 8kbit)

Realisierung mit 16k x 1 Bausteinen: Mindestens 8 Bausteine nötig. Die Speicherkapazität beträgt in diesem Fall 16kByte (= 128kbit)

Aus diesen Beispielen erkennt man leicht, wann welcher Bausteinorganisation der Vorzug zu geben ist:

- Bei großem Speicherbedarf ist die Einzelbit-Organisation vorzuziehen, da diese Bausteine aus Dekodiergründen eine hohe Bitanzahl pro Baustein aufweisen.
- Bei kleinerem Speicherbedarf sollte der Speicher so organisiert sein, daß er mit möglichst wenigen Bausteinen zu realisieren ist, im Idealfall hat ein solcher Speicherbaustein Speicherzellen in der vollen gewünschten Wortlänge, deren Information gleichzeitig an den Bausteinpins anliegen.

Über das Problem der Speicherorganisation hinaus geht die Frage, welcher Speichertyp im einzelnen Anwendungsfall verwendet werden soll. Hier ist eine kurze Übersicht über den heutigen Stand der Technik:

a) Schreib/Lesespeicher

(RAM="random access memory") sind Bauelemente, bei denen jede Speicherstelle adressierbar ist, d.h. es wird jeweils die Speicherstelle geschrieben, gelöscht oder gelesen, deren zugehörige Adresse an den Adreßeingängen des Bausteines anliegt. Die englische Bezeichnung ist irreführend. Sie heißt: "Speicher mit wahlfreiem Zugriff". Die gleiche Art des Zugriffs liegt beim ROM vor. Der Unterschied liegt also nicht in der Art des Zugriffs, sondern im Aufbau der Speicherzelle. Die Bezeichnung RAM wird hier aber trotzdem verwendet, da sie sich in Fachkreisen und der Literatur völlig eingebürgert hat. Es gibt zwei Typen von RAM: statische und dynamische. Für dynamischen RAM wird die Information in der Gate-Kapazität von MOS-Transistoren gespeichert. Da diese einen endlichen Ohmschen Widerstand und folglich einen Leckstrom aufweisen, muß die Ladung in gewissen Zeitabständen "aufgefrischt" werden, bevor der Schwellwert "logisch 1" unterschritten wird, damit die Information nicht zerstört wird.

Statische RAM verwenden Flipflops als Speicherzellen. Solange Betriebsspannung am Speicher anliegt, bleibt die Information erhalten, ohne daß Maßnahmen zur Auffrischung notwendig sind. Mit dem Abschalten der Versorgungsspannung geht allerdings in jedem RAM der Speicherinhalt verloren. Man bezeichnet sie daher als flüchtige (volatile) Speicher, im Gegensatz zu den ROM, die nicht flüchtig (non volatile) sind. Microcomputer, die RAM als Programmspeicher verwenden, benötigen daher stets Vorrichtungen, die ein Einlesen des Programms nach dem Einschalten der Betriebsspannung ermöglichen.

Der auffälligste Unterschied in der Organisation eines RAM im Vergleich zum ROM besteht darin, daß wegen der hinzugekommenen Schreibfunktion der Datenweg nicht mehr nur den Charakter eines Ausgangs hat, sondern bidirektional ist, also wahlweise als Aus- oder Eingang wirkt. In Bild 28 ist die Speicherorganisation eines 8 bit x 128 Worte RAM wiedergegeben.

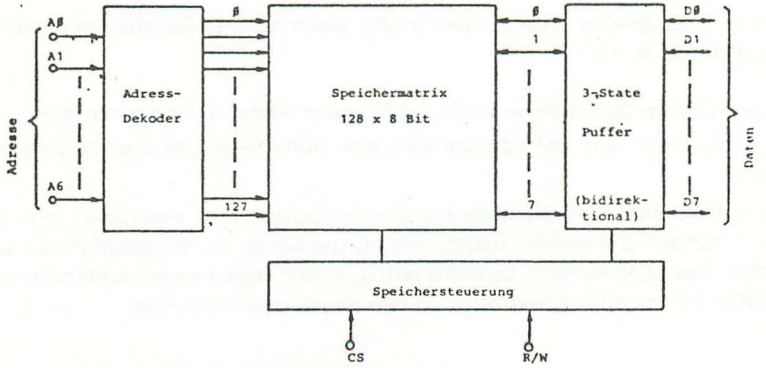


Bild 28: Organisation eines 128 x 8 bit RAMs

Außer den bidirektionalen Datenleitungen findet man noch eine Besonderheit: die Leitung R/W. Der logische Pegel auf dieser Leitung legt fest, ob der Speicher im Lese- oder Schreibbetrieb arbeitet.

Auch wenn die RAM in Mikrocomputern nicht immer als Programmspeicher-Verwendung finden, sind sie als Variablenspeicher und für Unterprogramm- und Interruptbehandlung durch Stapelspeicher grundsätzlich erforderlich.

b) Festwertspeicher

Das ROM (**R**ead **O**nly **M**emory), das seinen Namen zu Unrecht führt - es müßte richtiger Read-Only-RAM heißen -, ist ein frei adressierbarer Festwertspeicher, dessen Information dem Baustein vom Hersteller über die Metallisierungsmaske nach Kundenwunsch fest eingegeben wird; sie kann also bei Stromausfall nicht verlorengehen, ist aber auch nicht abänderbar.

In ROMs speichert man Programme, bei denen keinerlei Änderung zu erwarten ist. Wegen der Maskenprogrammierung ist eine Mindest-Bausteinabnahme Voraussetzung (etwa ab 1000 Stück).

Masken-ROMS sind die billigsten Halbleiter-Speicherbauelemente.
Das PROM (**P**rogrammable **ROM**) ist ein ROM, das man mit einem Programmiergerät selbst elektrisch programmieren kann. Das Programmieren erfolgt meist durch gezieltes Durchbrennen von Diodenstrecken. Änderungen sind nur insoweit möglich, als man nachträglich bisher nicht programmierte Bits "einbrennt".

Vorteile des PROMs: Der Anwender kann die Programmierung selbst vornehmen. Er braucht also sein Programm nicht aus dem Haus zu geben. PROMs sind in bipolarer Technik aufgebaut, daher sehr schnell und brauchen nur 1 Versorgungsspannung (+5V).
Nachteil: PROMs sind teurer als ROMs.

Das EPROM (**E**rasable **PROM**) oder REPROM (**R**e-programmable **ROM**) kann mit Hilfe von ultraviolettem Licht "en bloc" gelöscht und mit einem Programmiergerät wie ein PROM wieder neu programmiert werden.
Dadurch ergibt sich die Möglichkeit der Fehlerkorrektur und man kann Programmänderungen durchführen, ohne daß man die vorher programmierten Schaltkreise wegwerfen muß.
Nachteil: EPROMs sind noch teurer als PROMs und brauchen meist 3 Versorgungsspannungen (– 5V, 12V).

Das EEPROM (electrically erasable **PROM**) entspricht dem vorher beschriebenen EPROM, jedoch kann auch das Löschen auf elektronischem Wege (also nicht mit ultraviolettem Licht) erfolgen.

Dadurch ist ein gezieltes Löschen einzelner Speicherzellen möglich, ohne daß der Baustein aus dem Computer herausgenommen und "en bloc" gelöscht werden müßte.

Nachteile der EEPROM:s gegenüber EPROMs

- hoher Preis
- lange Zugriffszeit
- mehrere (typisch 4) Versorgungsspannungen für den Betrieb nötig.

2.8.2. Grundsätzliches zum Anschluß von Festwert- und Schreib/Lesespeichern

Der Mikroprozessor Z80-CPU ist so konzipiert, daß an ihn direkt (d.h. ohne irgendwelche zusätzliche Interface-Hardware) sämtliche gängigen statischen und dynamischen Speicherbausteine (ROMs, PROMs, EPROMs, EEPROMs, RAMs) anschließbar sind.
Aufgrund der Architektur solcher Speicherbausteine (Kompexität, Anzahl der Chipauswahl -Eingänge, Adreßmultiplex usw.) ihrer max. Zugriffzeit und der endlichen Strom- und Kapazitätsbelastbarkeit der CPU-Ausgänge (1,8 mA) können jedoch - je nach Gesamtgröße, Architektur und Typ des Speichers - folgende Schaltmaßnahmen nötig werden:

- Pufferung von Adreß- Daten- und Steuerbus der Z80-CPU
- Adreßcodierung zur Aktivierung der einzelnen Bausteine
- Seitenauswahl-Schaltung bei Speicherblöcken, deren Adreßbereichszuweisung umschaltbar ("jumper" - oder "switch-selectable") bleiben soll.
- Adreßmultiplexen bei Verwendung von Speicherbausteinen mit gemultiplexten Adreßleitungen (z.B. 16 pin - dynam. 16kbit-RAMs)
- Schaltung zur Erzeugung eines WAIT-Signals bei Verwendung langsamer Speicherbausteine.

Für diese aufgeführten Schaltungsmaßnahmen werden Beispiele gezeigt und darüberhinaus näher auf Timing-Fragen bei der Verwendung dynamischer RAMs eingegangen.

Zu beachten ist, daß sämtliche Mikrocomputerbausteine des Systems Z80 mit einer einzigen 5V-Versorgungsspannung auskommen, heute verfügbare EPROMs und EEPROMs jedoch mehrere Versorgungsspannungen benötigen.
Daher kann im System Z80 die Verwendung von "fusible" PROMs besonders vorteilhaft sein, die ebenfalls mit einer einzigen 5V-Spannung arbeiten (z.B. HARRIS 7641 oder 7781). Das Preisverhältnis zwischen "fusible" PROMs und EPROMs ist 1:2, so daß man bei der Verwendung von PROMs erst bei der dritten Version eines Festwertspeicherinhaltes höhere Kosten als bei der Verwendung und Neuprogrammierung von "UV"-EPROMs in Kauf nehmen müßte, wobei die zusätzlichen EPROM-Stromversorgungskosten noch nicht in die Rechnung einbezogen wurden.

2.8.2.1. Minimalkonfiguration

Obwohl das System Z80 aufgrund seiner außerordentlich hohen Leistungsfähigkeit und Verarbeitungsgeschwindigkeit nicht zur Implementierung von Klein-Systemen gedacht ist, soll hier eine Minimalkonfiguration gezeigt werden (Bild 29). Sie verwendet einen 2 kByte-Masken-ROM-Programmspeicherbaustein.

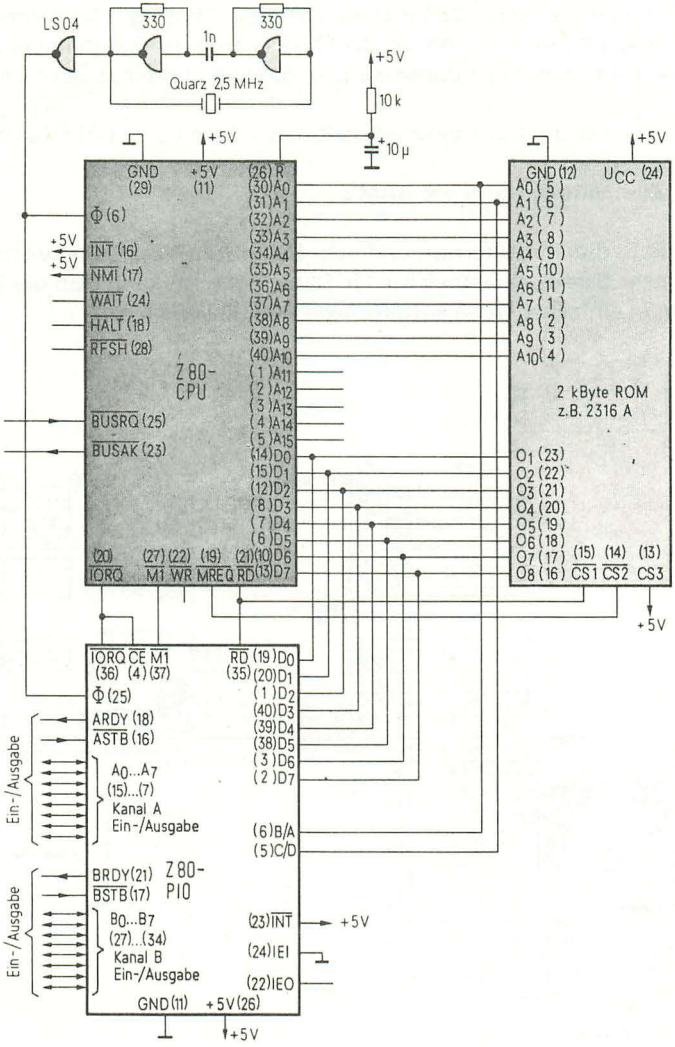


Bild 29: Minimalkonfiguration im System Z80

Externes RAM wurde nicht vorgesehen, d.h. Interrupt- und Subroutinenbehandlung ist mit dieser Schaltung nicht möglich, da das Stack im System Z80 in externem RAM liegt. Trotzdem ist der gezeigte Computer voll arbeitsfähig, da die Z80-CPU über einen internen zweiten Registersatz verfügt, der als RAM-Ersatz in beschränktem Umfang verwendet werden kann und die beiden Ports der Z80-PIO als Ein- und Ausgabe-Ports (incl. Handshaking) verwendbar sind. Die Adreßleitungen A_0 und A_1 wurden zur Adressierung der beiden Ports innerhalb des -Select) respektive zur Unterscheidung zwischen Steuerworte und -transfers/Daten (C/D-Select) herangezogen.

Beim Hinzufügen weiterer Z80-PIO im Abfrage (="Polling")-Verfahren oder anderer Ein/Ausgabebausteine können die CE-Eingänge der Bausteine natürlich nicht mehr direkt mit IORQ verbunden werden, sondern müssen geeignet decodiert werden.
Einzelheiten über die Ein/Ausgabe-Schaltungen finden Sie in Abschnitt 2.9.

2.8.2.2. Zuschaltung statischer RAM's

Für den Fall, daß mit Interrupt und/oder Subroutinen gearbeitet werden soll oder einfach mehr Schreib/Lesespeicher für Daten nötig ist, kann man die Minimalkonfiguration in einfacher Weise entsprechend Bild 30 erweitern.

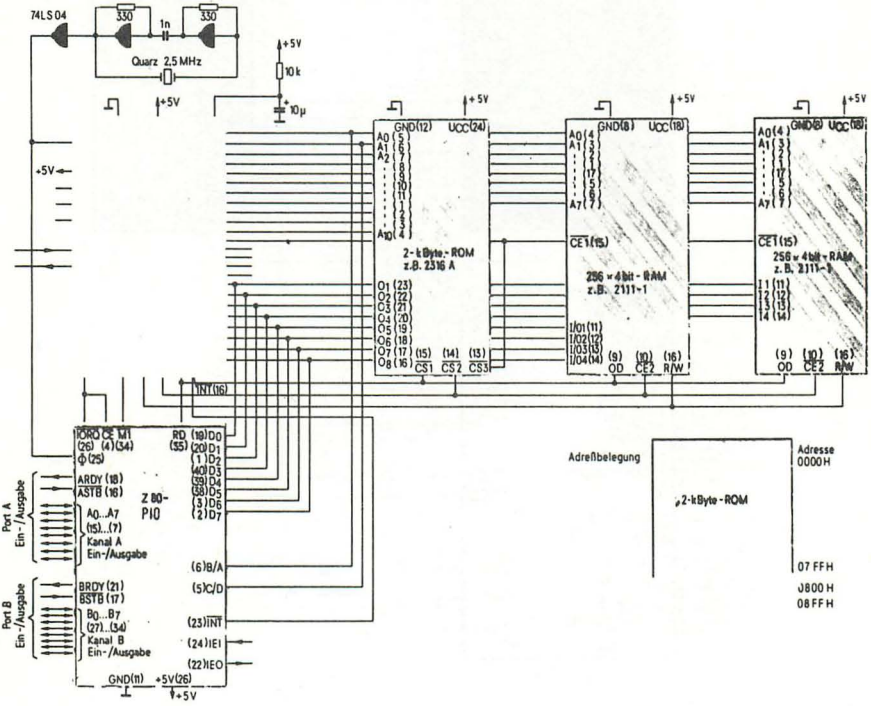


Bild 30: Durch zusätzliche RAM erweiterte Minimalkonfiguration

2.8.2.3. Realisierung eines voll ausbaufähigen Computers

Die nötige **Pufferung** im Fall des Anschlusses bipolarer Bausteine ergibt sich einfach aus der Summe der "Fan In" sämtlicher auf einer Leitung liegenden Anschlüssen; bei Pufferung bidirektionaler Signale muß darüberhinaus den Pufferbausteinen 'gesagt" werden, in welche Richtung die Datenübertragung momentan erfolgt.

Beim Anschluß von MOS-Speicherbausteinen ergibt sich die Grenze der höchstens anschließbaren Eingänge aus der kapazitiven Belastung, die diese Eingänge für die CPU (oder den Treiberbaustein) darstellen und die dadurch zusammen mit dem Ausgangsinnenwiderstand entstehende erhöhte Schaltzeit bzw. verminderte Flankensteilheit.

Zur **Bausteinauswahl** verwendet man Standardgatterbausteine, falls nur wenige "Chip-Select"-Signale erzeugt werden müssen, sonst 1 aus 8-Decoder. Es soll auch noch an die Möglichkeit erinnert werden, komplexere Decodierungen durch PROM's vorzunehmen; dadurch läßt sich in vielen Fällen die erforderliche Bausteinstückzahl drastisch senken.

Bild 31 zeigt eine solche voll ausbaubare Schaltung; die zugehörige Adreßbelegung geht ebenfalls aus Bild 31 hervor.

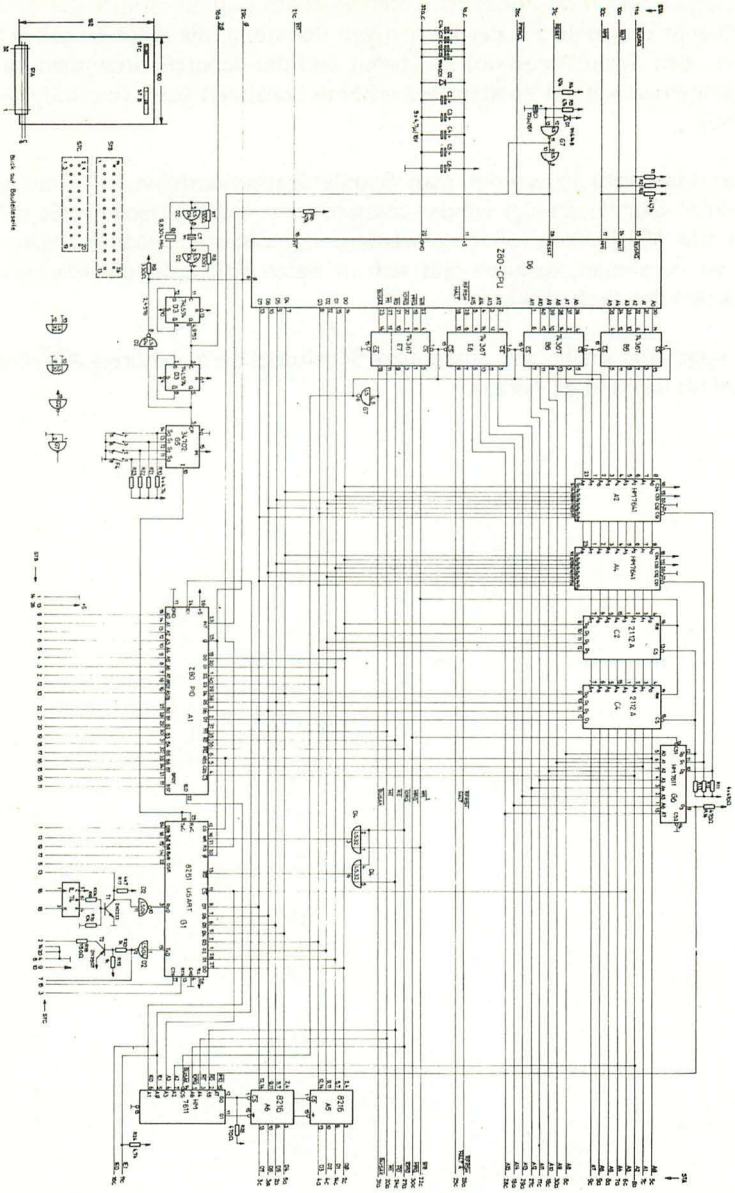
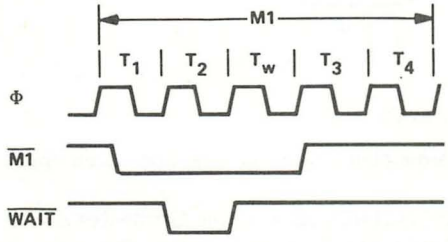


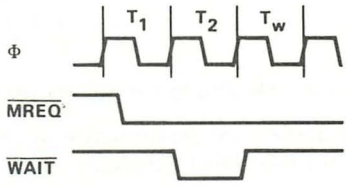
Bild 31: Komplette Z80-Mikrocomputer-Schaltung

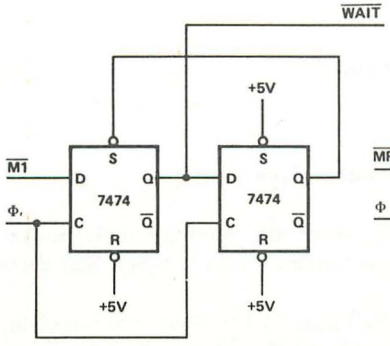
2.8.2.4. Anschluß "langsamer ' Speicherbausteine

Die meisten Speicherbausteine gibt es in mehreren Geschwindigkeitsversionen, wobei die Bausteine mit kürzerer Zugriffszeit häufig signifikant teurer sind als die etwas langsameren Versionen. Da umgekehrt in vielen Anwendungen ein gewisser Verlust von Verarbeitungsgeschwindigkeit der CPU tragbar ist, kann man in diesen Fällen durch Erzeugung sogenannter WAIT-Zyklen in den Genuß niedriger Speicherbausteinkosten kommen.
Hierzu ist lediglich nötig, aus den Steuersignalen der Z80-CPU ein Warte-Signal (WAIT) zu erzeugen, daß der Z80-CPU wiederum über den WAIT-Eingang zugeführt wird und diese veranlaßt, den nächsten Maschinenzyklus erst einen Taktimpuls später als standardmäßig zu beginnen.

Je nachdem, ob lediglich dem besonders zeitkritischen M1-Zyklus oder aber jedem Speicherzugriffszyklus ein "WAIT"-Zyklus T_W hinzugefügt werden soll, ist die Schaltung nach Bild 32 respektive nach Bild 33 zu verwenden; in den Bildern ist auch das entsprechende Zeitverhalten der Schaltung angegeben.







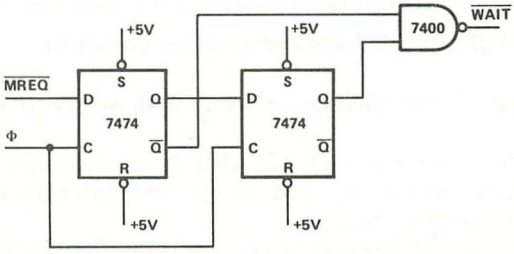


Bild 32: Verlängerung des Lese-Zyklus durch Einfügen eines WAIT-Signals (aus CPU-Manual)

Bild 33: Verlängerung eines jeden Speicherzugriffs Signals durch Einfügen eines WAIT-

2.8.2.5. Schaltung zur Speicherbereichauswahl

Bei modularem Aufbau von Mikrocomputern (d.h. in Fällen, wo die Gesamtschaltung aus einer Reihe von Standard-Steckkarten zusammengestellt wird) ist es wesentlich, die Adreßbereiche von Speicherkarten mit Schaltern (z.B. DIP-Switch) einstellbar zu halten. Eine einfache Schaltung hierfür zeigt Bild 34.

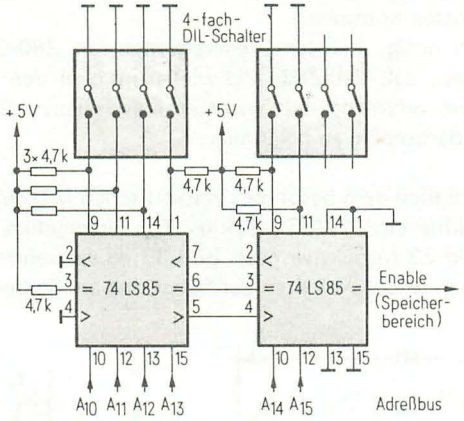


Bild 34: Schaltung zur Speicherbereichsauswahl

Sie besteht im wesentlichen aus 4 bit-Vergleichern und den Speicherbereichsauswahl (meist DIL)-Schaltern;
die Anzahl der nötigen Schalter und Komparatoren ist von der Größe des zu verschiebenden Speicherblocks und der Größe des Speicherbereichs abhängig, in dem der Speicherblock verschoben werden soll.

$$\text{Anzahl Schalter} = \text{ld}\left\{\frac{\text{Speicherbereich}^*}{\text{Speicherblockgröße}}\right\}$$

Anzahl Vergleicherbausteine = 1/4 {Anzahl Schalter}

2.8.3. Anschluß von dynamischen Speichern

2.8.3.1. CPU-Signale für den Betrieb von dynamischen Speichern

Der Mikroprozessor Z80-CPU ist so ausgelegt, daß er mit minimalem Hardware-Aufwand die Verwendung von standardmäßig lieferbaren dynamischen Speicherbausteinen erlaubt.
Die entsprechenden Steuersignale sind $\overline{\text{MREQ}}$, $\overline{\text{WR}}$ und $\overline{\text{REFR}}$. Beachten Sie bitte, daß die Adressen 125 ns vor der Aktivierung des MREQ-Signals stabil sind und die Information auf dem Datenbus 220 ns vor Aktivierung des $\overline{\text{WR}}$-Signals gültig ist.

* Der Speicherbereich ist in dieser Formel so zu wählen, daß er bei ganzzahligen Vielfachen der Speicherblockgröße beginnt und endet.

Bei der Verwendung von 4kbit-RAM's in 22 Pin-Gehäusen kann das Signal $\overline{MREQ}$ direkt als Steuerung des $\overline{CE}$-Takteingangs und das Signal $\overline{WR}$ zur Steuerung des $\overline{WRITE}$-Eingangs der Speicherbausteine verwendet werden.
Bei der Verwendung von 16 pin– 4kbit dynamischen RAM's wird das $\overline{MREQ}$-Signal zur Erzeugung des $\overline{RAS}$-Signals direkt verwendet, während Adreßmultiplexen und die Erzeugung des $\overline{RAS}$ - strobe - Signals durch zusätzliche Standard-Hardware erfolgen. Das CPU-Signal $\overline{WR}$ liegt wie bei den 22 pin-Bausteinen direkt am $\overline{WRITE}$-Eingang der Speicherbausteine an.
Der kritische Punkt für das Timing beim Anschluß von dynamischen RAM-Bausteinen ist der erste Maschinenzyklus $\overline{M1}$ ("fetch"-Zyklus). Hierbei geschieht der gesamte Speicherzugriff innerhalb der ersten 2 Taktimpulse. Man sieht, daß zwischen dem Zeitpunkt der Stabilisierung der Adressen und dem Signal $\overline{MREQ}$ 125 ns liegen, die bei weitem ausreichen, um die nötigen Decodier- und Einschwingzeiten abzudecken.
Kritischer ist die Situation in der High-Periode des $\overline{MREQ}$, deren Minimalzeit zu 170 ns garantiert ist.

Die speicherseitig minimal erforderliche Dauer dieses Signals (min. "prechargeduration") liegt bei 120 - 150 ns, da jedoch das $\overline{MREQ}$-Signal diverse Adreßcoder und Puffer durchläuft, muß beim Design besonders darauf geachtet werden, daß an dieser Stelle vernünftige Schaltzeitreserven verbleiben.
Während normalen Lesezyklen wird $\overline{MREQ}$ 650 ns aktiv, bevor die zu lesenden Daten gültig sein müssen.

Bei Schreibzyklen ist $\overline{MREQ}$ genauso lange aktiv, wie bei Lesezyklen, wobei die einzuschreibenden Daten noch etwa 160 ns nach Ende des $\overline{MREQ}$-Signals gültig sein müssen und $\overline{WR}$ noch etwa 400 ns nach Beginn des $\overline{MREQ}$-Signals inaktiv bleibt.
Daraus folgt, daß bei 22 pin 4 kbit-Speicherbausteinen des $\overline{WR}$-Signals direkt die $\overline{WRITE}$-Eingänge der Speicher ansteuern kann. Bei 16 pin-RAM-Bausteinen kann das invertierte $\overline{RD}$ anstelle von $\overline{WR}$ benutzt werden, wenn nur beim Schreibzyklus die gültige Dateninformation vor $\overline{CAS}$ verfügbar ist.

Bei Computerschaltungen, die auf einer einzigen Platine aufgebaut sind, kommt man mit einer einfachen Pufferung entsprechend Bild 35 aus. Dort sind auch die Verzögerungszeiten, die in einer solchen Schaltung zu berücksichtigen sind, angegeben; man sieht, daß 16 pin-dynamische RAM-Bausteine mit 400 ns Zugriffszeit bzw. 22 pin-Bausteine mit 380 ns Zugriffszeit in einem System mit Z80-CPU ohne weiteres bei einer CPU-Taktfrequenz von 2,5 MHz arbeitsfähig sind.

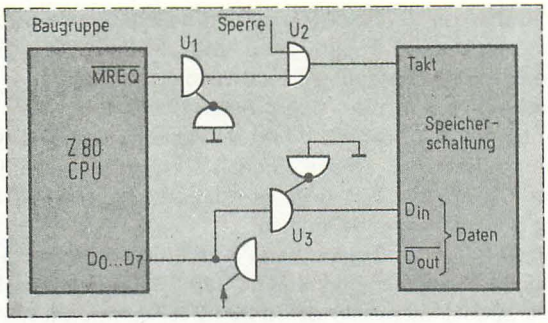


Bild 35: Prinzipschaltung einer "einfachen" Pufferung

Bei Computerschaltungen, die auf mehrere Baugruppen verteilt sind, ist es aus Störsicherheitsgründen empfehlenswert, eine zweifache Pufferung der Busleitungen vorzunehmen, d.h. je einen Satz Pufferbausteine auf der Baugruppe der Zentraleinheit und auf den einzelnen Speicher-Baugruppen vorzunehmen (Bild 36). In diesem Fall werden selbstverständlich höhere Anforderungen an die Zugriffszeit der Speicherbausteine gestellt. 22 pin-Bausteine dürfen eine Schreib/Lese-Zykluszeit von höchstens 315 ns haben, während 16 pin-Schaltkreise mit max. 340 ns auskommen. Sorgfältige Leistungsführung ist dabei selbstverständlich; insbesondere sollte dafür gesorgt werden, daß die Baugruppen mit den dynamischen RAM's möglichst nahe an der Zentraleinheits-Platine gelagert sind.

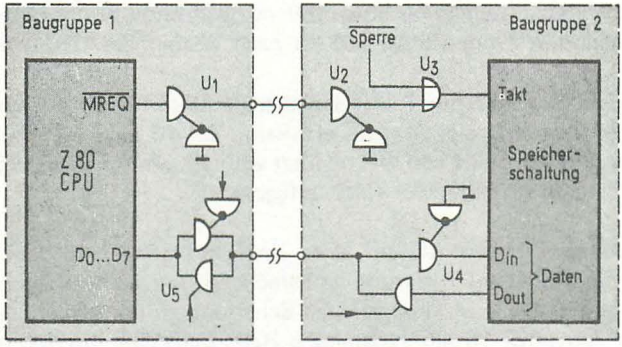


Bild 36: Prinzipschaltung einer "doppelten" Pufferung

Bei einer CPU Taktfrequenz von 2,5 MHz erfolgen Speicherzugriffe im Mittel alle 1,6 μsec.

Obwohl bei der Verwendung eines 2,5 MHz-Taktes Refresh etwa 9 mal öfter als notwendig (2 Millisekunden) erfolgt, würde durch Beschränkung auf die minimal nötigen Refresh-Zyklen an Leistungsverbrauch nur 100 Milliwatt eingespart werden. In Fällen, wo minimaler Leistungsverbrauch besonders wichtig ist, wird man daher mit Hilfe von Timer-Schaltungen eine Beschränkung der Refresh-Zugriffe auf das absolute Mindestmaß durchführen.

2.8.3.2. Grundsätzliche Hinweise zum Anschluß dynamischer RAM-Bausteine

Sollen andere als die in den bisherigen gezeigten Schaltungen verwendete Bausteine eingesetzt werden, müssen eine Reihe grundsätzlicher Regeln beachtet werden, da selbst pinkompatible dynamische RAM-Bausteine verschiedener Hersteller sich nicht exakt gleich verhalten.

a) Allgemeine Hinweise bei Verwendung von 22 pin-4kbit dynam. RAM-Bausteinen

- Das gesamte Timing der Bausteine ist vom $\overline{CE}$-Takt abgeleitet. Dieses Signal (MOS-Eingang) hat einen Hub von V_{dd}-V_{ss} und wird im allgemeinen mit einem TTL/MOS-Pegelwandlergatter erzeugt; die Verzögerungszeit eines solchen Pegelwandlers liegt in der Größenordnung von 50 ns.
  Der Speicherbaustein zieht praktisch nur Strom, solange das $\overline{CE}$-Signal aktiv ("HIGH") ist.

- Adressen und das Signal $\overline{CS}$ müssen vor der steigenden Flanke von $\overline{CE}$ eingeschwungen, d.h., daß zu diesem Zeitpunkt sämtliche Anwahlvorgänge abgeschlossen sein müssen. $\overline{CS}$ und die Adreßinformationen werden im allgemeinen intern zwischengespeichert und dürfen nach einer vorgegebenen Zeit wieder ungültig werden.
  Des weiteren kann bei manchen Bausteinen der Leistungsverbrauch dadurch reduziert werden, daß man sämtliche Adreßeingänge auf einen bestimmten Logikpegel bringt, in vielen Fällen ist übrigens diese Tatsache in den Datenblättern nicht erwähnt.

- Während des Lesezyklus muß der $\overline{WRITE}$-Eingang inaktiv (high) sein, und zwar vor der steigenden Flanke von $\overline{CE}$, um zu verhindern, daß die Information in den Speicherbausteinen zerstört wird. Während Schreibzyklen kann das $\overline{WRITE}$-Signal vor oder auch nach Beginn dieses Zyklus aktiv werden, wobei selbstverständlich grundsätzlich gewisse Restriktionen zu beachten sind.

- Der Ausgangspuffer ist vor und nach jedem Zyklus in hochohmigem Zustand. Er enthält nach der durch den Zugriff bedingten Verzögerungszeit die aus dem Speicher ausgelesenen Daten (während eines Lesezyklus, falls $\overline{CS}$ low ist) und geht nach Ende des $\overline{CE}$-Signals wieder in den hochohmigen Zustand über. An dieser Stelle soll noch angemerkt werden, daß man wegen ihrer geringeren Verzögerungszeit Tri-State-Treibern gegenüber Open-Collector-Treibern den Vorzug geben sollte; so würde sich beispielsweise bei einer Lastkapazität von 80 - 120 pF und einem 2,2 kOhm Pull-up-Widerstand eine Verzögerungszeit von etwa 100 ns ergeben.
- Die Eingangsspannungspegel von 22 pin 4 kbit-MOS-Bausteinen V_{il} und V_{ih} sind von Baustein zu Baustein unterschiedlich. In der Regel ist V_{il} gleich 0,6 V, während V_{ih} zwischen 2,2 und 3,0 V garantiert wird. Zur Sicherstellung eines Störabstandes von etwa 200 mV sowohl im high- als auch im low-Pegel müssen Treiberbausteine der Serie 74 oder 74 H mit Pull-up-Widerständen verwendet werden, so daß auch Bausteine mit Vih = 2,2 V oder mehr den gewünschten Störabstand erreichen.

- Während $\overline{CE}$ inaktiv ist, werden die internen Kapazitäten der Speicherschaltkreise vom $\overline{MREQ}$-Signal (high) aufgeladen. Die kritischsten Zeitverhältnisse treten, wie schon angedeutet, während des M_1-Zyklus auf, und zwar zwischen dem Befehls-Lese-Zyklus (fetch-Zyklus) und dem Refresh. Da als Dauer des Signals $\overline{MREQ}$ der Z80-CPU 170 ns minimal garantiert sind und die entsprechenden Zeitbedingungen der dynamischen RAM's in der Regel bei 120 - 150 ns liegen, muß dafür Sorge getragen werden, daß sämtliche Puffer-, Decoder- und Verdrahtungslaufzeiten unter der restlichen verfügbaren Verzögerungszeit bleiben. In kritischen Fällen darf daher an dieser Stelle auch die Temperaturabhängigkeit der Laufzeiten nicht vernachlässigt werden (genügende Laufzeitreserve!).
- Da die $\overline{CE}$-Eingänge der Speicherbausteine relativ große Eingangskapazitäten haben (25 - 30 pF) ist beim Schaltungsdesign besonders große Sorgfalt auf die Erzeugung dieses $\overline{CE}$-Signals zu legen. Da in einem 8 bit-System mindestens 8 solche $\overline{CE}$-Eingänge an einem $\overline{CE}$-Takttreiber angeschlossen sind, kann vor allem durch ein unzureichendes Platinen-Layout eine Laufzeitkette entstehen, in der die Qualität des am letzten Baustein anlangenden $\overline{CE}$-Signals nicht mehr ausreichend ist.
- Die Versorgungsspannungen V_{dd} ($\pm$ 12V $\pm$5%) und V_{bb} (–5V +5%) werden von $\overline{CE}$ bausteinintern ein- und ausgeschaltet, dadurch entstehen bei den $\overline{CE}$-Flanken Hochfrequenzstörungen auf diesen Speiseleitungen, die besonders sorgfältig über Hochfrequenz-Kondensatoren (Keramik) möglichst dicht an den V_{dd} und V_{bb} Anschlüssen gegen Masse (V_{ss}) kurzgeschlossen werden müssen.
  Darüberhinaus sollte man versuchen, den Speicherbereich auf der Leiterplatine so kompakt wie irgend möglich zu machen und die Adreß- und Datenpuffer räumlich möglichst nahe an die Speicherbausteine legen.
- Die 22 pin 4 k dynamischen RAM's haben einen Leistungsverbrauch, der direkt proportional der Zugriffshäufigkeit und der "Ein"-Zeit des $\overline{CE}$-Taktes Aus diesem Grund kann man die Gesamtverlustleistung eines Speichersystems dadurch verringern, daß man $\overline{CE}$ decodiert und nur dem Speicherbereich zuführt, der momentan tatsächlich aktiv ist.

b) Allgemeine Betrachtungen zum Einsatz von 16 pin 4kbit dynam. RAM-Bausteinen

Bei diesen Bausteinen wird die Adresse über 6 Adreßleitungen eingelesen, wobei diese 6 Adreßleitungen über 2 Taktsignale in Zeilen- und Spaltenadresse gemultiplext werden.
Im Gegensatz zu den 22 pin 4 kbit dynamischen Bausteinen sind die 16 pin-Bausteine verschiedener Hersteller fast in allen Fällen risikolos austauschbar. Trotzdem soll hier noch eine Liste der wichtigsten Gesichtspunkte bei der Verwendung der 16 pin Bausteine gegeben werden:

- Jeder Zugriffszyklus beginnt mit dem Zeilen-Adreß-Takt $\overline{RAS}$ (row-adress-strobe), der sämtliche Zeilenadressen von den Adreßleitungen einliest und

zwischenspeichert. Dabei wird eine der 64 Zeilen ausgewählt und sämtliche 64 Speicherstellen auf die zugehörigen Leseverstärker geschaltet, die die Informationen aufnehmen, zwischenspeichern und wieder in die Zelle einschreiben. Auf diese Weise erfolgt durch $\overline{RAS}$ ein von $\overline{CAS}$ völlig unabhängiger Refresh.

- Der zweite Teil eines jeden Zugriffs wird vom Spaltenadreßtakt $\overline{CAS}$ ("Column adress strobe") eingeleitet, der die Spaltenadressen einliest und abspeichert. Die Signale $\overline{WRITE}$ und $\overline{CS}$ müssen nicht unbedingt bis zum Auftreten von $\overline{CAS}$ anliegen. Entsprechend dem Zeilenauswahltakt wird auch $\overline{CAS}$ decodiert, aktiviert einen von 64 Leseverstärkern/Zwischenspeichern und schaltet den ausgewählten Zwischenspeicher auf die Ausgangstreiberstufe.
  Bei den meisten lieferbaren 4 kbit-16 pin-Bausteinen werden die Daten in Zwischenspeicher geladen und bleiben gültig, bis der nächste $\overline{CAS}$-Impuls auftritt.

- Da der $\overline{RAS}$-Zyklus beginnen kann, sobald die Zeilenadresse stabil anliegt, jedoch bevor der Speicherbereichsauswahlvorgang abgeschlossen ist, sparen 16 pin RAM's gegenüber 22 pin RAM's etwa 50 - 100 ns. Da hier jedoch die Verlustleistung in erster Linie von $\overline{RAS}$ abhängt, wird man in den meisten Fällen RAS decodieren, wodurch ein Teil dieses Geschwindigkeitsvorteils verlorengeht.

- Die Eingangspegel von $\overline{RAS}$ und $\overline{CAS}$ liegen beim low-Pegel (V_{il}) zwischen 0,6 und 0,8 V, bei high-Pegel (V_{ih}) zwischen 2,4 und 2,7 V. Die Ansteuerung dieser Eingänge über TTL-Gatter erfolgt daher am besten unter Verwendung von Pull-up-Widerständen. Schottky-Treiberbausteine sollten bei Speichern mit V_{il} kleiner als 0,7 V nicht verwendet werden, da sie ein V_{o1} kleiner gleich 0,5 V haben und damit nicht der geforderte Störabstand von 200 mV erreichbar ist.
  Praktisch resultiert aus der Möglichkeit, bei 16 pin RAM's TTL-Treiber anstelle von TTL/MOS-Pegelwandlern zu verwenden, eine Zeitersparnis von 20 - 40 ns.

- Die 6 Adreßleitungen müssen zwischen Zeile und Spalte nach der Eingabe der Zeilenadresse umgeschaltet werden. Die Kapazitäten der Adreßeingänge liegen bei 5 - 10 pF, so daß ein 16 kByte-Speicherbereich für den Treiber eine Last von 150 - 250 pF pro Adresse darstellt. Da die meisten TTL-Ausgänge nur für kapazitative Belastung unter 50 pF spezifiziert sind, muß man entweder einen Bustreiber, wie 74S50 oder 7428 (für 150 pF Lastkapazität spezifiziert) verwenden.
  Wegen des Eingangspegels V_{il} = 2,4 V, den die meisten Bausteine haben, ist zur Erreichung eines ausreichenden Störabstandes ein Pull-up-Widerstand erforderlich.

- Der $\overline{WRITE}$-Eingang wird erst abgefragt, wenn das $\overline{CAS}$-Signal aktiv wird. Wird $\overline{WRITE}$ vor CAS aktiv, so führt das RAM einen Schreibzyklus in der kürzestmöglichen Zykluszeit aus. Wird $\overline{WRITE}$ erst nach Beginn von $\overline{CAS}$ aktiv, so führt das RAM einen Lese/Änderungs-Schreibzyklus durch.

- Bei einem Schreibzugriff werden die eingehenden Daten durch $\overline{CAS}$ oder WRITE übernommen, und zwar von dem Signal, das als letztes auftritt. Man kann somit durch Verzögerung von $\overline{WRITE}$ gegenüber CAS erreichen, daß die Information erst nach Auftreten des $\overline{WRITE}$-Signals übernommen wird.
- Der Ausgangspuffer der 16 pin Schaltkreise speichert die ausgelesenen Daten zwischen, sobald $\overline{CAS}$ aktiv wird und während $\overline{CS}$ aktiv ist. Erscheint ein Signal ohne $\overline{CS}$, werden die Ausgangspuffer/Zwischenspeicher gesperrt und
- Die Ausgänge können wired-or-verknüpft werden, soweit $\overline{CAS}$ bei Auswahl eines Speicherbereiches sämtlichen Bausteinen zugeführt wird, während RAS decodiert und nur dem angesprochenen Speicherbereich zugeführt ist.
  Sobald ein CAS-Signal empfangen ist, gehen sämtliche Ausgangstreiber der Speicher in den hochohmigen Zustand über, so daß keine Buskonflikte auftreten können.
- Refresh wird ausschließlich über das $\overline{RAS}$ erzeugt ($\overline{CAS}$-Signal nicht nötig) und hat alle 2 ms zu erfolgen. Dies läßt sich für DMA-Anwendungen ausnutzen, bei denen das Refresh-Signal der CPU während des M1-Zyklus unterbunden wird. Man läßt dann eine Serie von 64 aufeinanderfolgenden Zugriffen zu, die alle 2 ms lediglich RAS erzeugen. Dadurch wird ein transpartenter Refresh durchgeführt.
- Da die interne Schaltung dynamisch aufgebaut ist, muß bei jedem Einschalten der Versorgungsspannung eine Reihe von "leeren" $\overline{RAS}/\overline{CAS}$-Zyklen erzeugt werden, um eine definierte Vor-Aufladung zu garantieren. Auch sollten $\overline{RAS}$-Zyklen allein nicht über einen längeren Zeitraum als 2 ms gegeben werden, da sonst die Ausgangstreiber unter Umständen unkorrektes Verhalten zeigen. Entsprechend sollten auch in einem solchen Fall mehrere "leere" $\overline{CAS}$-Zyklen gegeben werden, bevor Daten von den Bausteinen ausgewertet werden.
- Genauso wie bei den 22 pin-Bausteinen ist bei den 16 pin-Bausteinen die Verlustleistung proportional der Anzahl der Zugriffe pro Zeiteinheit. Anders als bei den 22 pin-Bausteinen ist jedoch bei vielen 16 pin-Bausteinen der Leistungsverbrauch nicht von der Dauer der einzelnen Zugriffe abhängig, da hier in sämtlichen internen Schaltungen der Bausteine konsequent dynamische Speicherzellen verwendet werden.

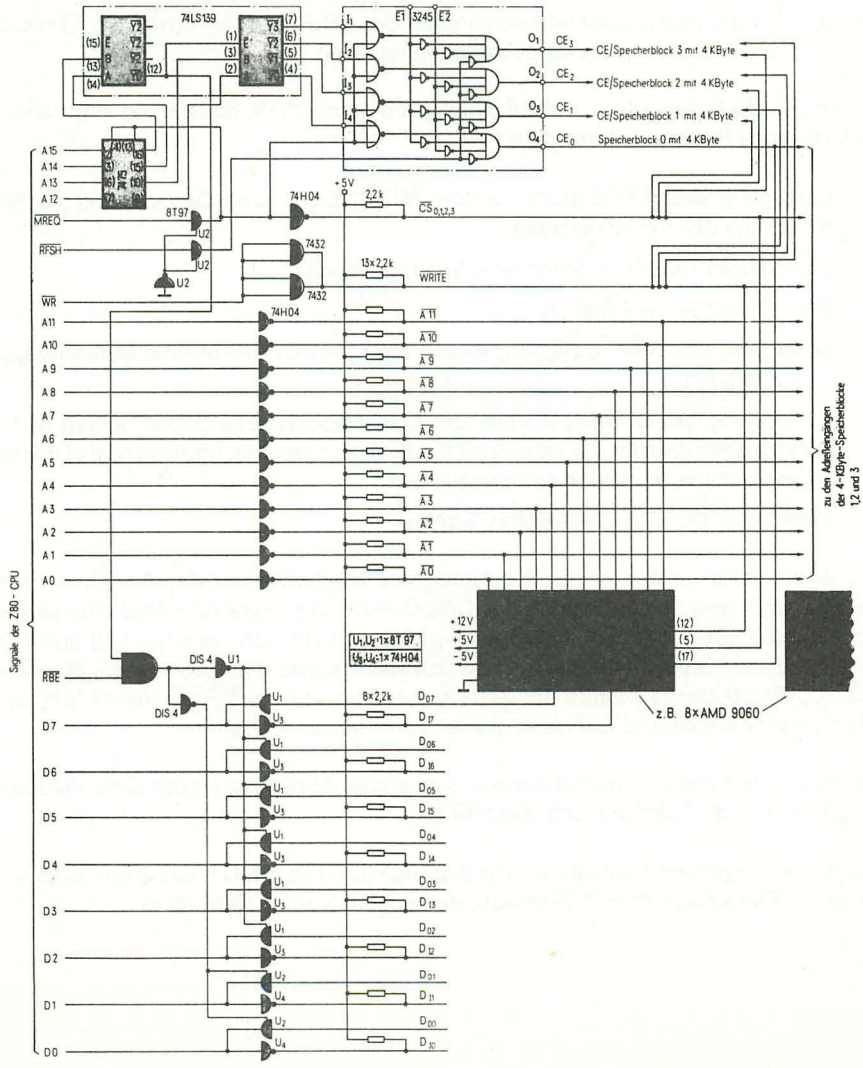
2.8.3.3. Schaltung mit 22 pin dynamischen 4 kbit-RAM-Bausteinen

Die beschriebene Schaltung arbeitet mit praktisch jedem beliebigen 22 pin dynamischen RAM-Baustein, bei dem der Zeitabstand zwischen dem Beginn des $\overline{CE}$-Signals und dem Zeitpunkt, bei dem die Information auf dem Datenbus gültig sind, kleiner als 350 ns ist, wobei für Pufferbausteine und Verdrahtungen eine Gesamtverzögerung von 30 ns übrigbleibt.

Wie erwähnt, treten die zeitkritischsten Bedingungen während des M_1-Zyklus auf. Bei einer Betriebsfrequenz von 2,5 MHz ist die minimale Zeit in diesem Lesezyklus 440 ns. Dabei müssen selbstverständlich die Verzögerungszeiten der Steuer- und Decodierschaltungen berücksichtigt werden.

Der obere Teil von Bild 37 zeigt die Schaltung eines solchen Systems. Ein Decoder führt die Seiten - (''page'')-Auswahl durch, ein zweiter ist für das Umschalten zwischen den vier $\overline{CE}$-Takten verwendet. Die Adreßleitungen A 12 - A 15 sind den Speicherbausteinen über einen getakteten Pufferspeicher zugeführt, um eventuelle Störungen auf den Adreßleitungen während des $\overline{MREQ}$-Signals aufzufangen; diese Maßnahme ist bei den Adreßleitungen A0-A11 nicht nötig, da diese bereits in den Speicherbausteinen intern zwischengespeichert sind.

Bild 37: Komplette Schaltung mit 22pin-dynamischen 4k x 1bit – RAM-Bausteinen



Das WRITE-Signal für die Speicher wird direkt aus dem WR-Signal des Z80-CPU erzeugt; es wurden Treiberschaltkreise mit Pull-up-Widerständen zwischengeschaltet, um einen ausreichenden Störabstand (V_{il} = 0,4 V und V_{ih} = 2,6 V) zu erreichen.
Aus dem gleichen Grunde wurden auch Adreß- und Datenbus mit Treiberstufen mit Pull-up-Widerständen gepuffert.

2.8.3.4. Schaltung mit 16 pin dynamischen 4 kbit-RAM-Bausteinen

Die im folgenden gezeigte Schaltung arbeitet mit praktisch jedem 16 pin 4 kbit-dynamischen Speicherbaustein, der einen Lese/Schreibzugriffszyklus von kleiner als 350 ns hat, wobei Laufzeiten zwischen der CPU und den Speichern (Treiber, Verdrahtung) als kleiner als 40 ns angenommen sind.

In einem Speichersystem mit 16 pin-dynamischen RAM-Bausteinen sind besonders folgende Punkte zu beachten:

- Erzeugung des $\overline{\text{RAS}}$-Signals aus dem $\overline{\text{MREQ}}$-Signal und Decodierung zur Minimierung der Verlustleistung.
- Umschalten zwischen Zeilen- und Spalten-Adresse
- Erzeugung des $\overline{\text{CAS}}$-Signals
- Erzeugung des $\overline{\text{WRITE}}$-Signals, sobald die Information auf dem Datenbus gültig ist ($\overline{\text{WRITE}}$ kann vor oder nach $\overline{\text{CAS}}$ beginnen).
- Verbindung des $\overline{\text{CAS}}$-Signals mit sämtlichen Speicherbausteinen, soweit deren Datenausgangspuffer wired-or-verknüpft sind, um sicherzustellen, daß nichtangesprochene Bausteine gesperrt bleiben.
- Erzeugung des $\overline{\text{RAS}}/\overline{\text{CAS}}$-Refres-Signals.

Die Bilder 38 a ... 38 c zeigen 3 verschiedene Möglichkeiten des Adreßmultiplexens (MUX) und der Erzeugung des CAS-Signals. Die einfachste Methode ist die Verwendung einer digitalen Verzögerungskette mit Anzapfungen bei 125 und 175 ns (Bild 38 a). Diese Methode ist natürlich recht aufwendig. Eine zweite Möglichkeit ist, die steigende Flanke des Φ-Impulses, der auf das $\overline{\text{MREQ}}$-Signal folgt, als MUX zu verwenden und $\overline{\text{CAS}}$ über Gatterlaufzeiten zu erzeugen.

Bei dieser Methode entsteht allerdings eine starke Abhängigkeit der CAS-Verzögerungszeit von der Taktfrequenz (Bild 38 b).

Die dritte Möglichkeit verwendet ein 2Φ-Signal, erzeugt MUX aus Φ und $\overline{\text{CAS}}$ der fallenden Flanke von dem 2Φ-Impuls; der auf MUX folgt (Bild 38 c).

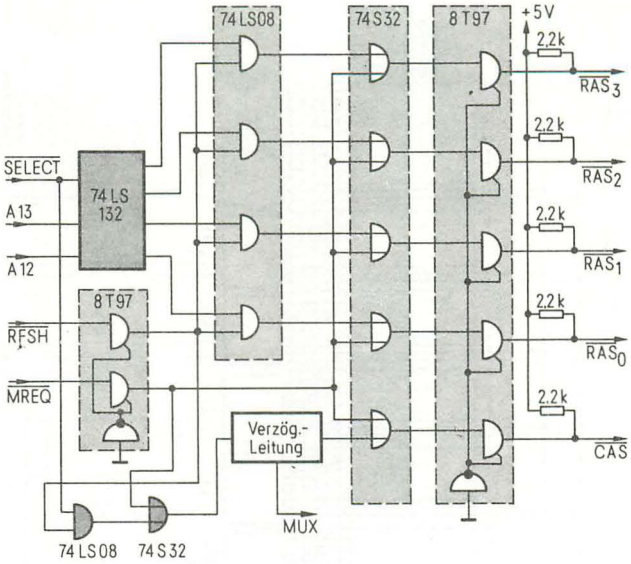


Bild 38a: Adreßmultiplexen (MUX) und Erzeugung des CAS-Signals mit einer angezapften digitalen Verzögerungskette

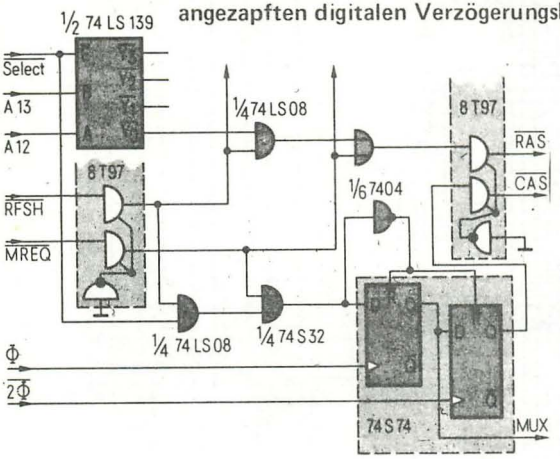


Bild 38c: Adreßmultiplexen und Erzeugung des CAS-Signals unter Ausnutzung der fallenden Flanke des 2 Φ-Impulses

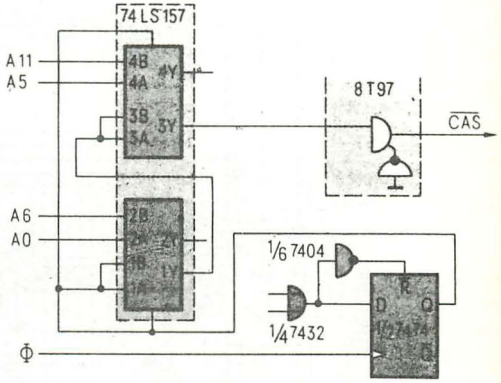


Bild 38b: Adreßmultiplexen und Erzeugung des CAS-Signals über Gatterlaufzeiten

Bild 39 zeigt eine vollständige Schaltung mit 16 pin RAM-Bausteinen. Das Adreßmultiplexen erfolgt durch abwechselnde Freigabe zweier Tri-State-Treiber. Der Vorteil dieser Methode ist, daß dabei die RAM-Adreß-Eingangskapazitäten während dem Zeilen/Spaltenumschalten wesentlich schneller umgeladen werden als dies bei der Verwendung normaler TTL-Treiberschaltkreise möglich wäre.

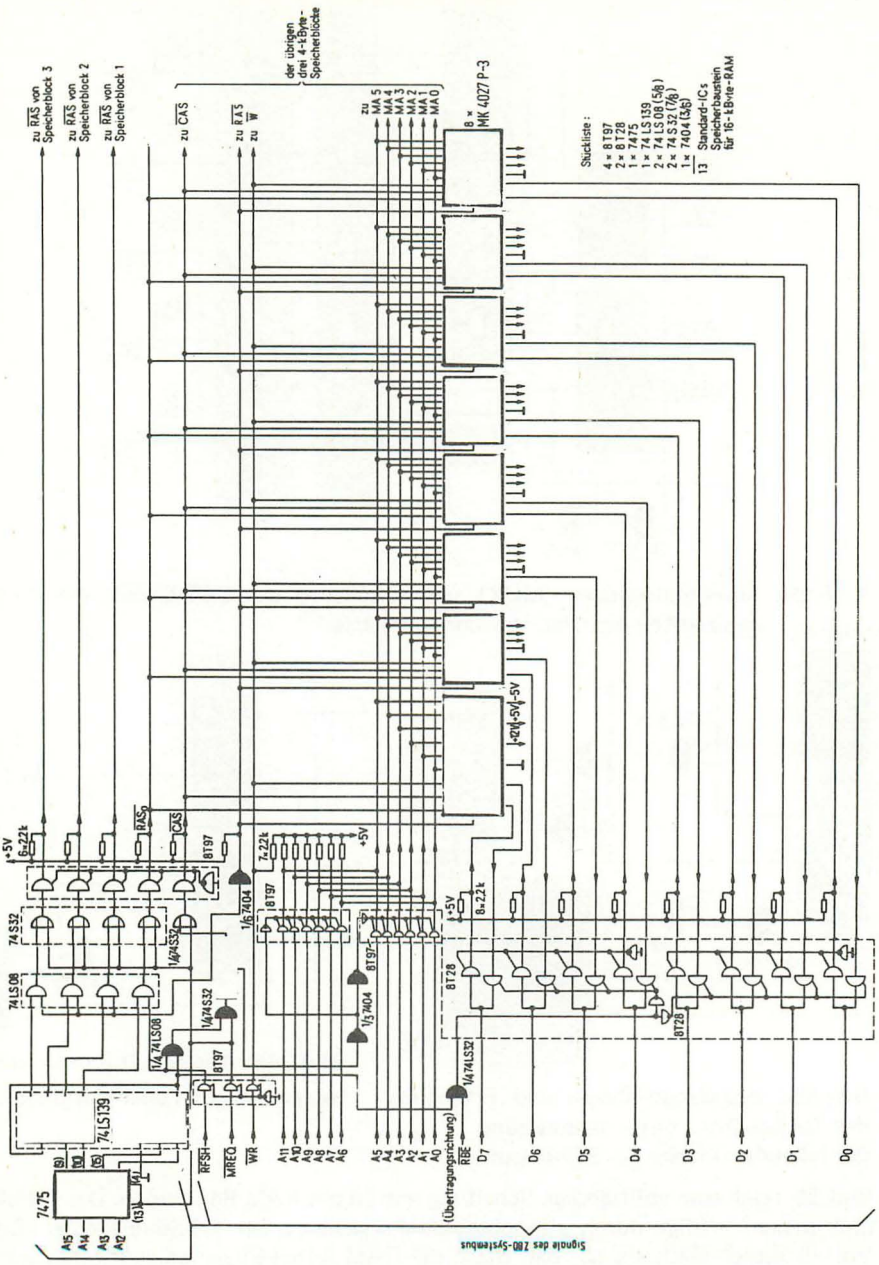


Bild 39: Komplette Schaltung mit 16pin-dynamischen 4k x 1bit RAM-Bausteinen

2.9. Ein/Ausgabe

Im vorliegenden Abschnitt sollen nun Fragen der Ein/Ausgabe und der Interrupt-Behandlung beim System Z80 ausgeführt werden.

2.9.1. Interrupt-Architektur

Im System Z80 wurde auf minimalen Schaltungsaufwand bei möglichst hoher Leistungsfähigkeit geachtet. Dies gilt insbesondere für die Interrupt-Behandlung, deren hohe Flexibilität und Leistungsfähigkeit weit über das bei Mikrocomputern übliche hinausgeht und dem Z80 Möglichkeiten eröffnet, die bisher teuren Prozeßrechnern vorbehalten waren. So verfügt die Z80-CPU über zwei grundsätzlich getrennte Interrupt-Behandlungsweisen:

a) Der "nichtmaskierbare Interrupt", bei dem die Interruptanforderung über einen hardwaremäßig separaten Eingang $\overline{\text{NMI}}$ an die CPU gelangt, und bei dem die Anfangsadresse der zugehörigen Interrupt-Bedienroutine fest auf der Programmspeicheradresse 0066 H liegt. Es soll hier angemerkt werden, daß beispielsweise das System 8080A, auf dem das System Z80 softwaremäßig aufbaut, über einen solchen gesonderten Interrupt nicht verfügt. Dieser nichtmaskierbare Interrupt wird von Interrupt-Freigabe-Flip-Flop der Z80 CPU nicht beeinflußt, d.h. eine Interrupt-Anforderung an diesen Pin läßt sich **nicht** durch einen "Interrupt- Disable"-Befehl per Programm sperren.
Sinn dieser Einrichtung ist, Alarmfälle in möglichst kurzer Zeit behandeln zu können (das ist deshalb gegeben, weil die Verzweigung auf die Interrupt-Bedienroutine hard- statt softwaremäßig realisiert ist), die vom momentanen Zustand der CPU und des Anwenderprogramms unabhängig sind (Beispiele: kritische Bedingung in einem Prozeß oder Stromausfall).
Dieser Interrupt hat grundsätzlich höchste Priorität.

b) Der "normale" Interrupt (Eingang INT), der durch die Befehle "Interrupt Disable" und "Interrupt Enable" per Anwenderprogramm zu sperren bzw. freizugeben ist. Hierbei ist das Verzweigen auf verschiedenen Interrupt-Bedienroutinen und das Priorisieren aller Interrupt-Anforderungen möglich.
Aus Kompatibilitätsgründen zum System 8080A verfügt die Z80-CPU über folgende drei Betriebsarten ("Interrupt modes"), die sich bei Programmbeginn durch den Befehl "Set Interrupt Mode" =

IM mode
(mit mode = 0, 1 oder 2)

vorwählen lassen.

- Mode 0, der genau der Interrupt-Behandlungsweise der 8080 A entspricht. Dabei sind in gängigen Hardwareimplementierungen maximal 8 Interrupt-Bedienroutinen möglich, deren Startadressen im Bereich 0 - 56 (dezimal) liegen und über die eine "RESTART" - Bedingung auf dem Datenbus ange-

sprungen werden können. Die Priorisierung muß durch gesonderte Hardware erfolgen (z.B. Prioritäts-encoderbausteine).
Wegen seiner eingeschränkten Möglichkeiten wird man dieses Interrupt-Behandlungsverfahren in Z80-Systemen nur verwenden, wenn bereits vorhandene 8080A-Programme auf Z80-Hardware unverändert ablaufen können sollen.

- Mode 1 erlaubt nur die Verwendung einer einzigen Interruptebene ("Restart 7").
  Ähnlich wie Mode 0 dient auch Mode 1 zur Herstellung der Kompatibilität zu 8080A; er entspricht der Verwendung des 8228 (mit 1kΩ gegen + 12V am "INTA"- Pin) als Sender einer einzigen Restart Adresse (0038 H)

- Mode 2 beinhaltet die für das System Z80 charakteristische Betriebsart, die eine praktisch unbegrenzte Anzahl an Interruptebenen zuläßt, wobei die gesamte Priorisierung in den Z80-Ein/Ausgabebausteinen ohne zusätzlichen Schaltungsaufwand geschieht.

Der entsprechende Informationsfluß ist in Bild 40 schematisch dargestellt:
Vor dem erstmöglichen Auftreten eines Interrupts (üblicherweise zu Programmbeginn nach folgendem Schema:

```
DI  ;  SICHERSTELLUNG DEFINIERTER
   ;  INTERRUPT – VERHÄLTNISSE
   ;  ES FOLGEN SÄMTLICHE CPU – UND
   ;  I/O – INITIALISIERUNGEN

IM 2;  INTERRUPT – MODE = 2

EI  ;  JETZT KANN INT FREIGEGEBEN WERDEN
```

um definierte Verhältnisse beim Einschalten zu vermeiden) werden CPU und Peripherie initialisiert.

Dabei ist zweierlei zu tun:

a) in das I-Register der Z80-CPU (8bit) wird der höherwertige Teil einer 16-Bit-Adresse (Interrupt-Service-Routinen-Zeiger) geladen.
b) in das Vektor-Register jedes Z80-Peripheriebausteins wird der gewünschte niederwertigere Teil einer 16-bit-Adresse (Interrupt-Service-Routinen-Zeiger) geladen.

Wie aus Bild 40 ersichtlich, sind die Interrupt-Anforderungsausgänge "$\overline{\text{INT}}$" sämtlicher Peripheriebausteine (Open-Drain-Ausgänge) miteinander und mit dem Interrupt-Anforderungseingang $\overline{\text{INT}}$ der Z80-CPU verbunden. Aufgrund der internen Prioritätsfestlegung (siehe einige Zeilen später) kann nur der $\overline{\text{INT}}$-Ausgang aktiv (low) werden, dessen Port die momentan höchste Priorität hat. Es ist also eindeutig definiert, welches Port eine Interrupt-Anforderung ausgestellt hat.

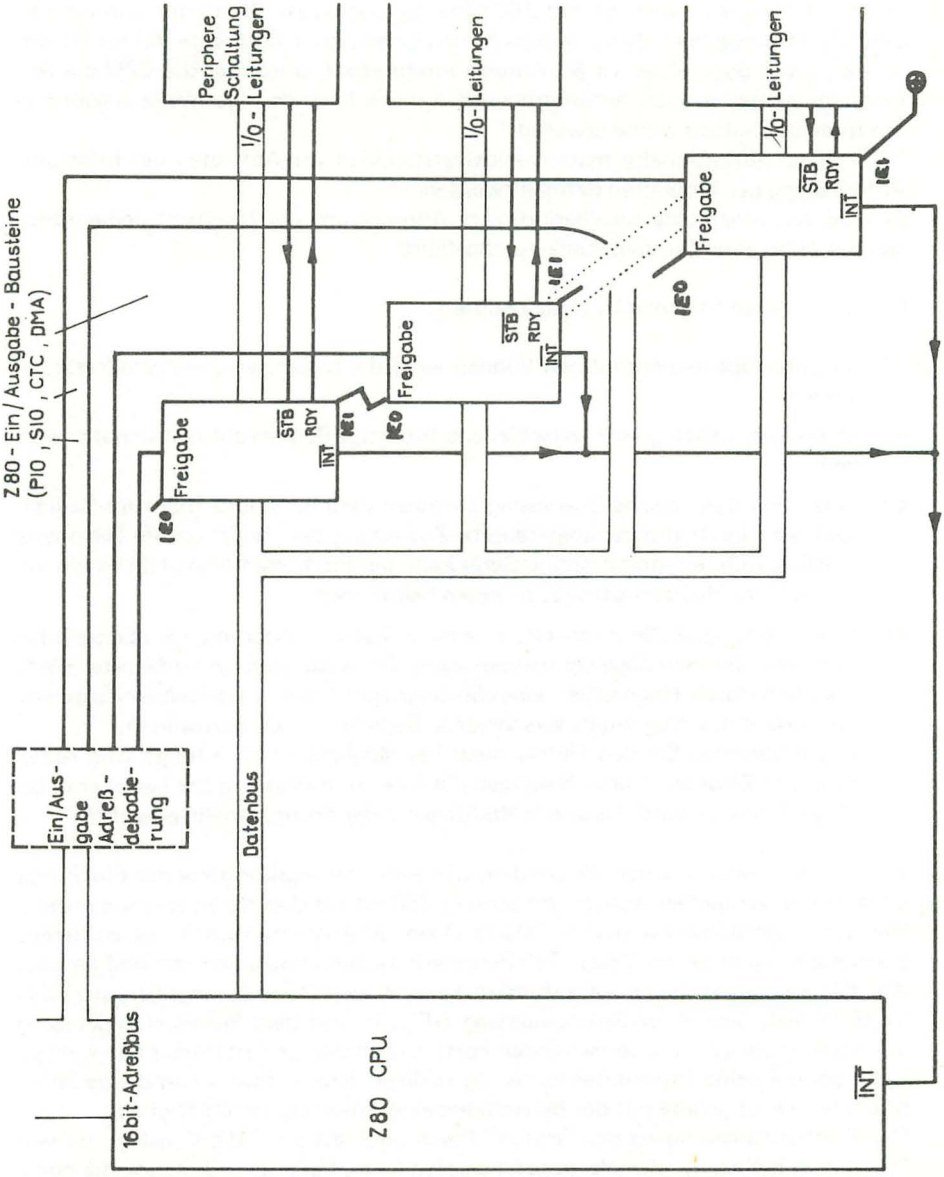


Bild 40: Z80-Interrupt-Architektur

Eben dieses Port legt nun die in seinen anfangs geladenen Vektorregister gespeicherte untere Adressenhälfte auf den Datenbus.
Von der CPU wird dieser Wert gelesen, sobald die Interrupt-Anforderung der Peripherie eingetroffen und der momentan in Ausführung befindliche Befehl abgearbeitet ist.

Der zu Anfang ins I-Register der Z80-CPU abgespeicherte Wert wird nun von der CPU als höherwertiges Byte, der von der Peripherie her eingelesene Vektor als niederwertigeres Byte einer 16 Bit-Adresse interpretiert, unter der die CPU die Anfangsadresse der von der anfordernden Peripherie in eindeutiger Weise zugeordneten Interruptbedienroutine erwartet.
Auch diese Adreßinhalte müssen selbstverständlich vor Auftreten der Interrupt-Anforderung per Programm definiert werden.
Es wird also eine echte speicherindirekte Adressierung der Interrupt-Bedienroutinen mit Hilfe einer Sprungtabelle durchgeführt.

Dadurch werden folgende Vorteile realisiert:

a. die Interrupt-Bedienroutinen können auf jeder beliebigen Speicheradresse beginnen.

b. es können beliebig viele verschiedene Interrupt-Bedienroutinen benutzt werden;

c. es ist eine dynamische Zuweisung zwischen Peripherie und Interrupt-Bedienroutinen durch programmgesteuerte Zuweisung der Zeiger an die Peripherie möglich (d.h. ein und dasselbe Gerät kann per Programm einmal diese, ein andermal jene Bedienroutine zugewiesen bekommen).

d. da die Sprungtabelle wahlweise in einem Festwert- oder in einem Schreib-Lesespeicher-Bereich abgelegt werden kann, ist es möglich, in einfachster Weise (nämlich durch Hochzählen eines Sprungzeigers) ein und derselben Peripherie im Verlauf des Programms verschiedene Bedienroutinen zuzuweisen.
(Bitte beachten Sie den Unterschied: bei Möglichkeit c.) erfolgte eine Änderung der Zuweisung über Neuladen des Interruptvektors in der Peripherie, bei Möglichkeit d.) wird das durch Modifizieren der Sprungtabelle erreicht).

Es muß jetzt noch klargestellt werden, wie Anforderungskonflikte der Ein/Ausgabebausteine vermieden werden. Im System Z80 erfolgt dies durch eine sogenannte Hardware-Priorisierungskette (= "Daisy chain priority interrupt"). Sie ist bereits standardmäßig in jedem Zilog -Z80-Peripheriebaustein implementiert und bewirkt, daß das Port einen Interrupt anfordern kann, dessen Priorisierungs-Eingang High ist (Bild 40). Sein Priorisierungsausgang (IEO) ist mit dem Priorisierungseingang des nächstniedriger zu priorisierenden Ports verbunden, und ist immer dann =high, wenn gerade keine Interruptanforderung vorliegt, bzw. = low, wenn dieses höher priorisierte Port gerade mit der Behandlung eines Interrupt beschäftigt ist.
Der Priorisierungseingang des "ersten" Ports wird fest auf "High" gelegt, da sein Priorisierungseingang niemals gesperrt werden kann, hat er grundsätzlich die höchste Priorität; beinhaltet ein Z80-Baustein mehrere Ports (Z80-PIO, -CTC, -SIO) so ist die Priorität zur Einsparung von Pins chipintern festgelegt. Ein Beispiel hierfür finden Sie in Bild 41.

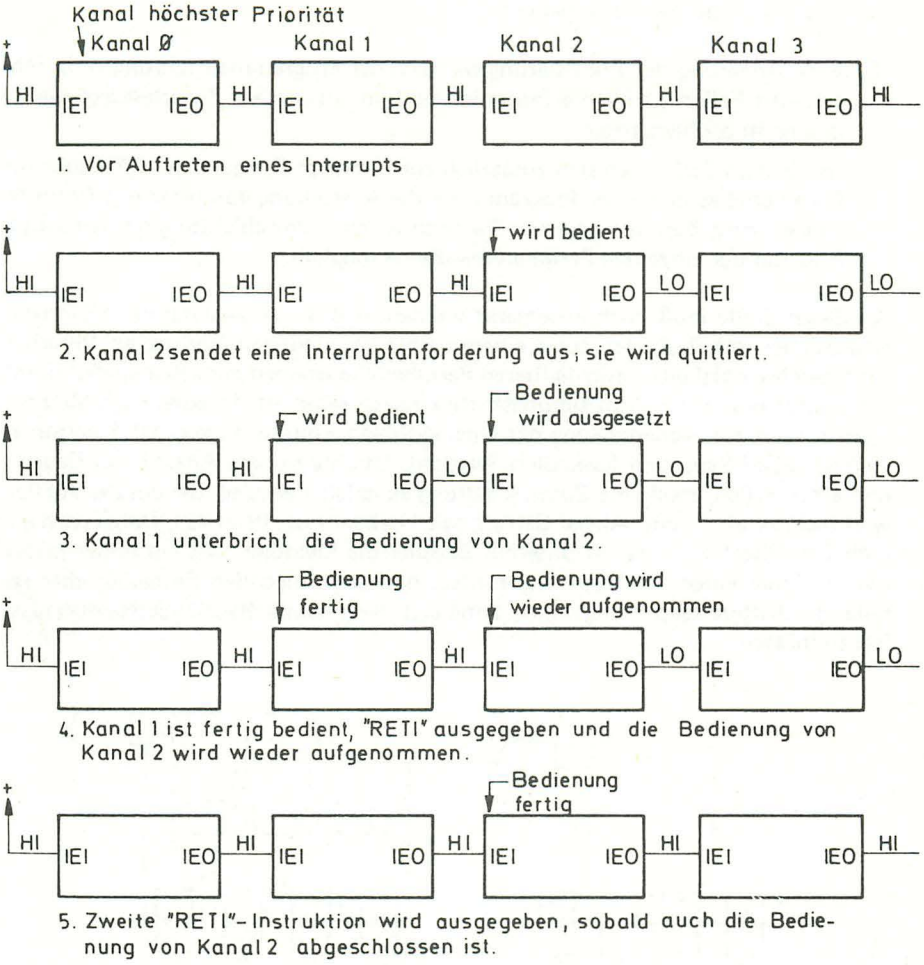


Bild 41: Interrupt-Priorisierung

Gegenüber einer softwaremäßigen Priorisierung hat diese Methode folgende Vorteile:

a. die gesamte Interrupt-Priorisierungs-Hardware verschwindet, da sie in den Ein/Ausgabe-Bausteinen des Systems untergebracht ist, ein "Interrupt-Controller" und ein spezieller Priorisierungsbaustein wird überflüssig.

b. es ist eine schnelle Interrupt-Bedienung möglich, da die höchste Priorität nicht erst von der CPU ermittelt werden muß.

Die hardwaremäßige Festlegung der Priorität wird manchmal als unangenehme Restriktion empfunden.

Dazu ist folgendes zu bemerken:

- ☐ Eine Änderung der Priorisierung während der Programmausführung ist in sehr seltenen Fällen wirklich erforderlich und im übrigen aus Zuverlässigkeitsgründen recht problematisch.
- ☐ Im System Z80 lassen sich zusätzlich zur Interrupt-Freigabe der CPU auch die Peripheriebausteine per Programm für die Ausstellung von Interrupt Anforderungen freigeben und sperren. Dadurch ist eine Umschichtung der Interrupt-Priorität der einzelnen Peripheriebausteine möglich.

An dieser Stelle muß noch angemerkt werden, daß die Anzahl der ohne Einfügen zusätzlicher WAIT-Zyklen (und einem damit verbundenen Verlust an Verarbeitungsgeschwindigkeit) kaskadierbaren Peripheriebausteinen zunächst einmal durch die Laufzeiten in der Priorisierungskette eingeschränkt ist. In einem 2,5 MHz-System sollten zur Sicherstellung der einwandfreien Funktion maximal 4 periphere Z80-Standard-Bausteine kaskadiert werden). Ist eine höhere Anzahl von Bausteinen erforderlich, muß eine Zusatzschaltung eingeführt werden, die aus der Addierwerktechnik als "Look Ahead Carry Logic" bekannt ist. (Bild 42). Dabei wird einfach der Übertrag (bzw. in unserem Beispiel die Meldung, daß ein höher priorisierter Kanal einen Interrupt angefordert hat) den folgenden Einheiten über parallel geschaltete Gatter zugeführt, ohne erst die gesamte Bausteinkette durchlaufen zu müssen.

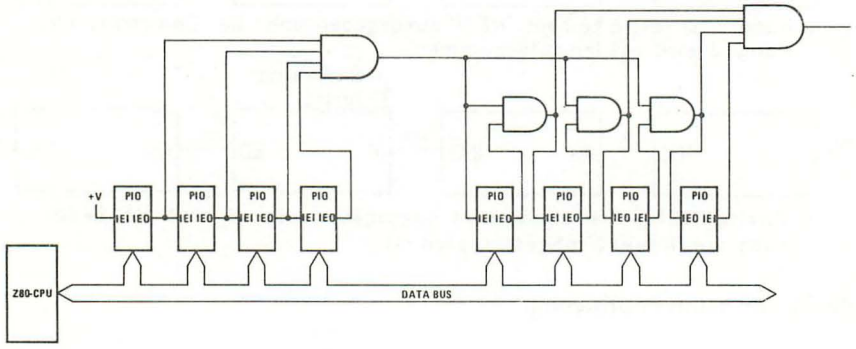


Bild 42: Look Ahead Carry Logic

2.9.2. Programmierung der Parallel-Ein/Ausgabe (Z80A-PIO und Z80-PIO)

Sämtliche Funktionen des Bausteins werden über Befehle des Z80 - Anwenderprogramms festgelegt.
Zum Verständnis der Funktionsweise ist die Kenntnis der Architektur (Bilder 43 bis 45) nötig.

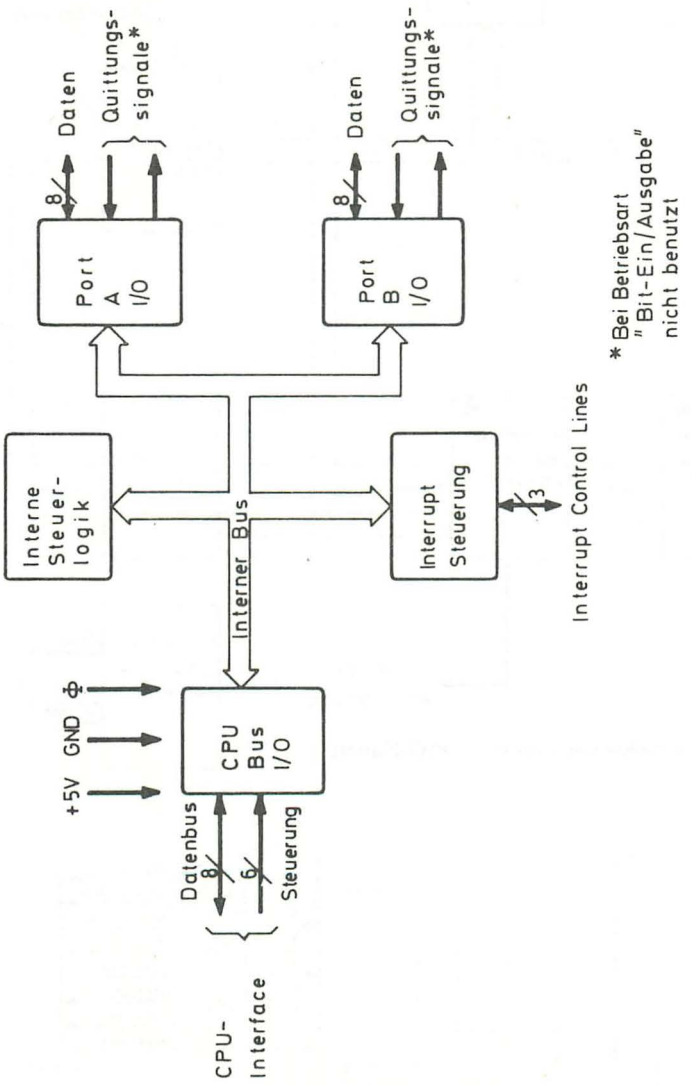


Bild 43: Blockschaltbild der Z80-PIO

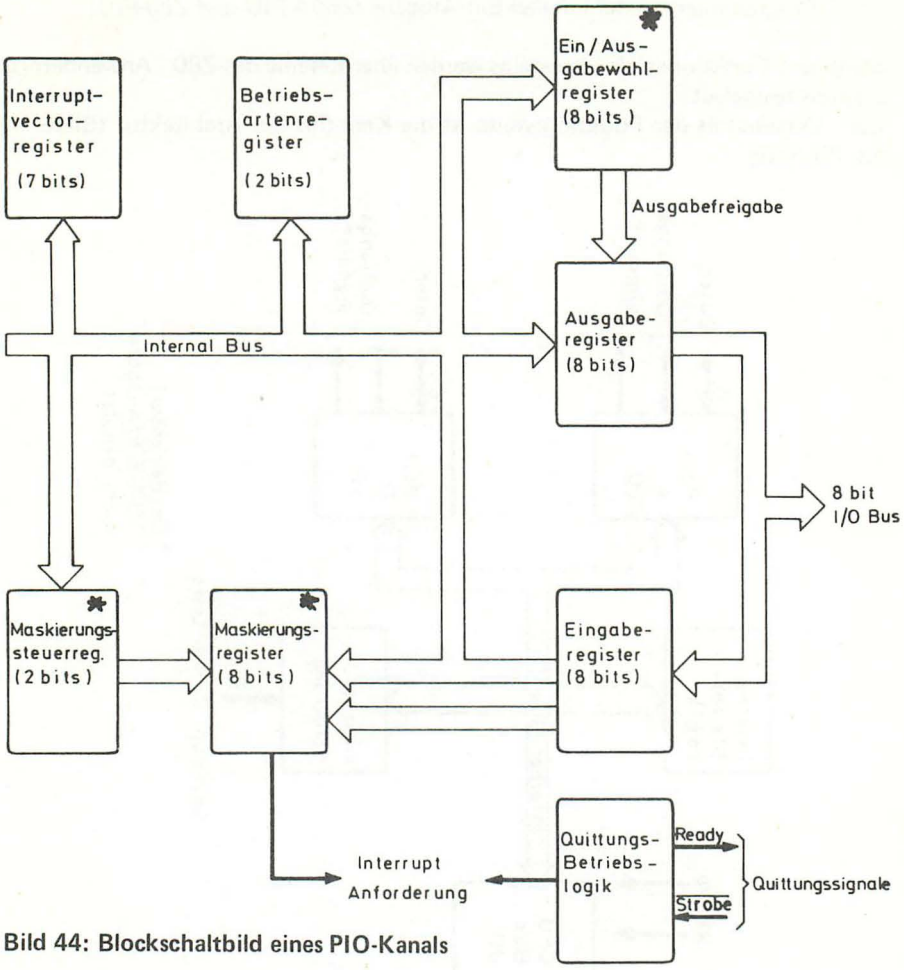


Bild 44: Blockschaltbild eines PIO-Kanals

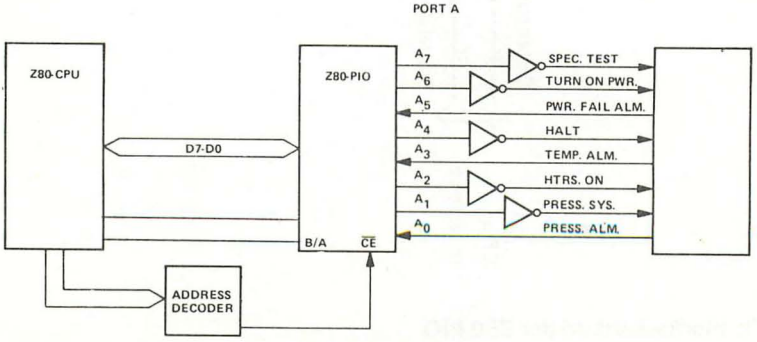


Bild 45: Adressierung von PIO-Steuer- und Datenpfad

Man sieht hieraus folgendes:

a. Ein Baustein Z80-PIO besteht im wesentlichen aus zwei 8bit-Ports, deren Funktion (Byte-Ausgabe, Byte-Eingabe, Byte-Ein/Ausgabe, Bit-Ein/Ausgabe) per Programm festgelegt wird und je zwei Quittungsleitungen je Port, die für die Interrupt-Abwicklung maßgeblich sind. Die beiden Ports werden automatisch über den Adreßeingang Pin 6 (= "Port A/Port B-Select") unterschieden, der im Allgemeinen mit A0 des Adreßbus verbunden wird.

b. Jedes Port wird über zwei Adressen angesprochen. Die Unterscheidung dieser beiden Adressen erfolgt automatisch über das Pin 5 ("Control/Data-Select"), das im allgemeinen mit A1 des Adreß-Bus verbunden wird.

c. Der Eingang $\overline{CE}$ sperrt einen ganzen Baustein bzw. gibt ihn frei. Er wird im allgemeinen aus den niederwertigen 8bit des Adreßbusses über einen Decoder abgeleitet (während C/D-Select und A/B-Select direkt mit A 1 bzw. A0 des Adreßbus verbunden sind (s. oben) und repräsentiert sämtliche 4 Adressen des PIO - Bausteins, nämlich

– Steuerwort-Pfad	von Port A	(B/A-SEL = Low C/D-SEL = Low)
– Daten-Pfad	von Port A	(B/A-SEL = Low C/D-SEL = High)
– Steurwort-Pfad	von Port B	(B/A-SEL = High) C/D-SEL = Low)
– Daten-Pfad	von Port B	(B/A-SEL = High C/D-SEL = High)

Über die Steuerwort-Pfade erfolgt die Übergabe der Steuerworte an die Ports über die üblichen Z80-Ausgabe-Befehle:

OUT (n), A
OUT (C), r
OUTI
OTIR
OUTD oder
OTDR

Folgende Steuerwörter können oder müssen dem Z80-PIO übergeben werden:

Auswahl der gewünschten Betriebsart

Hierfür wird das 2 Bit-Betriebsartauswahl-Register über die höchstwertigen 2 Bits (M1 und M0) eines Steuerworts, das von der CPU an das betreffende Port ausgegeben wird, gesetzt.
Format dieses Steuerworts:

D7	D6	D5	D4	D3	D2	D1	D0
M1	M0	X	X	1	1	1	1

M1 M0: Betriebsart
X X: nicht benutzt
1 1 1 1: identifiziert das Steuerwort als Betriebsartenauswahl-Wort

Betriebsart	M1	M0
Byte-Ausgabe	0	0
Byte-Eingabe	0	1
Byte-Ein/Ausgabe	1	0
Bit-Ein/Ausgabe	1	1

Laden des Interrupt-Vektors

Vor Auftreten der ersten Interrupt-Anforderung muß der jeweils gewünschte Vektor von der CPU in den PIO durch Angabe eines Steuerwortes an das betreffende Port mit folgendem Format geschrieben werden:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

identifiziert das Steuerwort als Interrupt-Vektor

Sondereinrichtungen bei der Betriebsart Bit-Ein/Ausgabe

In dieser Betriebsart ist nach der Ausgabe des Betriebsart-Auswahl-Steuerwortes von der CPU ein weiteres Steuerwort auszugeben, das die einzelnen Anschlüsse des betreffenden Ports als Eingänge bzw. als Ausgänge definiert.
Eine "1" in diesem Steuerwort bedeutet, daß der zugehörige PIO-Anschluß einen Eingang darstellt, eine 0 bedeutet, daß der Anschluß ab dann als Ausgang benutzt werden kann.

Format:

D7	D6	D5	D4	D3	D2	D1	D0
I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0

Interrupt-Steuerung

Bit	Wert	Bedeutung
7	0	Interruptflipflop rückgesetzt, Interrupt-Anforderungen werden nicht angenommen
7	1	Interruptflipflop gesetzt, Interrupt-Anforderungen werden bearbeitet
6,5,4	X	Werden nur bei Betriebsart Bit-Ein/Ausgabe benutzt, in allen anderen Betriebsarten ignoriert
3,2,1,0	0111	identifizieren das Steuerwort als Interrupt-Steuerwort

Format:

D7	D6	D5	D4	D3	D2	D1	D0
Interrupt Freigabe	UND/ ODER	High/ Low	nächstes Steuer-wort ist Maske	0	1	1	1
	nur in Betriebsart 3 benutzt			bedeutet Interrupt-Steuerwort			

Falls in diesem Steuerwort das Bit D4 = high war (="nächstes Steuerwort ist Maske"), muß ein weiteres Steuerwort an das Port ausgegeben werden.

Sein Format ist:

D7	D6	D5	D4	D3	D2	D1	D0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

Die Maske wirkt so, daß nur Anschlüsse mit Maskierungsbit MB_n=0 zur Erzeugung einer Interrupt-Anforderung herangezogen werden. Das Interrupt-Freigabe-Flip-Flop allein ist durch folgendes Steuerwort zu beeinflussen:

D7	D6	D5	D4	D3	D2	D1	D0
Int Freigabe	X	X	X	0	0	1	1

Bei "Int-Freigabe" = 1 wird die Interrupt-Anforderung einer peripheren Schaltung bearbeitet, bei "Int-Freigabe" = 0 wird eine Interrupt-Anforderung ignoriert.

Programmierungshilfen für die Z80-PIO

Laden des Interrupt-Vektors

V7	V6	V5	V4	V3	V3	V1	0

Betriebsart festlegen

M1	M0	X	X	1	1	1	1

M1	Mo	Betriebsart
0	0	Byte-Ausgabe ("O" wie "Output")
0	1	Byte-Eingabe ("I" wie "Input")
1	0	Byte-Ein/Ausgabe ("2" wie "2direktional")
1	1	Bit-Ein/Ausgabe (3)

Falls das Betriebsarten-Steuerwort für Bit-Ein/Ausgabe (="Control Mode, Mode 3". gesendet wurde, muß unmittelbar darauf folgendes Steuerwort zum Festlegen des I/O-Registers an den PIO gegeben werden:

I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

Wobei I/On = O das zu Bit Nr. n (n = o . . 7) gehörige Pin als Ausgabeanschluß definiert (..o" wie "Output") bzw.

I/O = 1 als Eingabeanschluß ("I" wie "Input").

Interrupt-Mode festlegen

Int Enable	AND OR	High/ Low	Mask/ Follows	0	1	1	1

Used in Mode 3 only

Nur in Betriebsart Bit-Ein/Ausgabe (3) benützt !

Soll mit diesem Steuerwort ausschließlich das Interrupt-Freigabe-Flip-Flop eines bestimmten Ports beeinflußt (gesetzt oder rückgesetzt) werden, kann **stattdessen**

folgendes Steuerwort gegeben werden:

INT Enable	X	X	X	0	0	1	1

Ist im obengenannten Interrupt-Steuerwort das Bit "Mask follows" (="nächstes Steuerwort ist Maske"), so muß unmittelbar auf dieses Steuerwort die Maske entsprechend folgendem Format folgen:

Ein Programmierbeispiel zeigt Bild 46. Hieraus ist übrigens ersichtlich, daß die Port-Adressen (Steuerwort und Datenpfad) bei Verwendung des residenten Z80-Assemblers in symbolischer Form angegeben werden können; selbstverständlich müssen ihnen vorher über die Pseudobefehle EQU oder DEFB Hexadezimalwerte zugewiesen werden.

Laden der PIO-Register (1 Port, nämlich Port A. Eine Datenübertragung würde danach durch den Befehl OUT (PAD,A) vorgenommen).

```
LD A, VECT

OUT (PAC),A

LD A, MODE3

OUT (PAC),A

LD A,I/0

OUT (PAC),A

LD   A, INTEIV

OUT (PAC),A

LD   A,MASK

OUT (PAC),A
```

Bild 46: Beispiel zur PIO-Programmierung

2.9.3. Programmierung der Zähler/Zeitgeber-Kanäle (Z80A-CTC und Z80-CTC)

Ebenso wie der Baustein Z80-PIO wird auch der Z80-CTC vom Anwenderprogramm gesteuert. Wir wollen auch hier zunächst kurz auf die Hardware-Architektur eingehen (Bild 47/48).

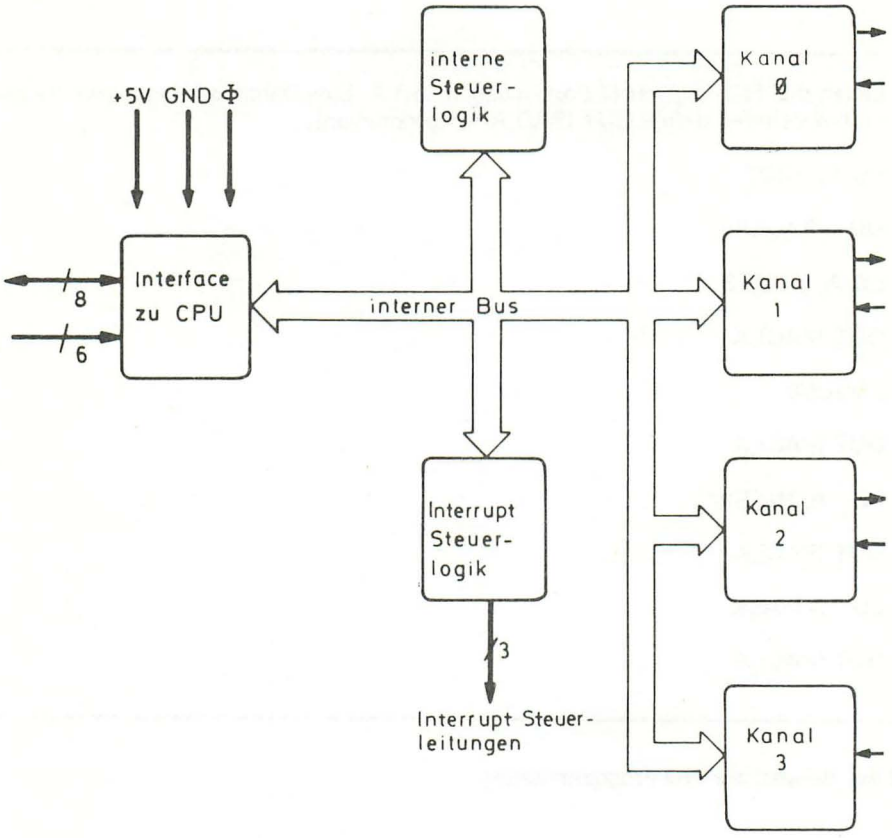


Bild 47: Blockschaltbild des Z80-CTC

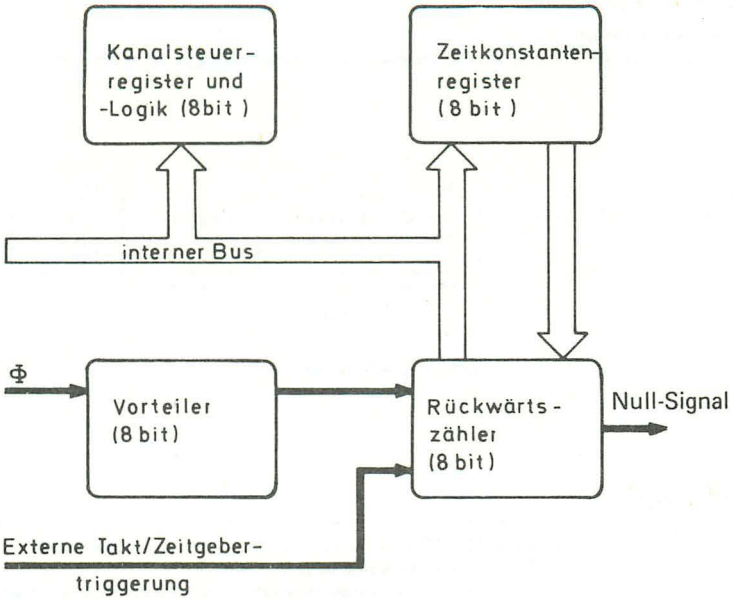


Bild 48: Blockschaltbild eines CTC-Kanals

Bekanntlich verfügt der Baustein über vier getrennte Zähler/Zeitgeberkanäle. Anders als beim Z80-PIO wird hier zwischen Steuerwörtern und Datenwörtern zum Setzen der Zeitkonstantenregister nicht durch ein eigenes "Control/Data"-Pin unterschieden, sondern durch ein vorangehendes Steuerwort. Das bedeutet, daß der Baustein Z80-CTC grundsätzlich ein Steuerwort erwartet, wenn er über Pin 16 (=$\overline{CE}$) freigegeben wird. Die Adressierung der vier verschiedenen Ports erfolgt über die beiden Pins 18 und 19 ("Channel Select 0" und Channel Select 1"). Grundsätzlich wird zwischen zwei verschiedenen Steuerworten unterschieden.

a) dem Steuerwort zur Betriebsartfestlegung, das durch eine "1" im niederwertigsten Bit gekennzeichnet ist. Einzelheiten finden Sie in Bild 49.

Bild 49: Programmierung des CTC

Festlegen der Betriebsart

Hat bit 0 den Wert 1, so wird das aktuelle Datenwort in das Kanalsteuerregister übernommen.

Format:

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
Interrupt Freigabe	Betriebs-art	Bereich Einheit	Trigger-flanke	Triggerzeit-punkt	Zeitkon-stante Laden	Rück-setzen	1

Nur für Betriebsart „Zeitgeber"

Bemerkung: Werte von D 0 ... D 6 sind beim Einschalten nicht definiert

Erläuterungen:

Bit.-Nr.	Wert	Erklärung
7	0	Interrupt des Kanals gesperrt
	1	Interrupt des Kanals freigegeben; Interruptanforderung erfolgt jedesmal, wenn der Rückwärtszähler den Wert Null erreicht. Durch ein Setzen des Bit 7 wird **kein** Interrupt durch ein **vorangegangenes** Zähler = Null-Ergebnis veranlaßt.
6	0	Betriebsart „Zeitgeber". Der Rückwärtszähler wird vom Vorteiler gestartet. Die Periode des Zählers ist dabei $c = t_c \cdot P \cdot TC$ mit t_c = Systemtakt-Periode P = Vorteiler (= 16 oder = 256) TC = 8 bit programmierbare Zeitkonstante (max. = 256)
	1	Betriebsart „Zähler". Der Rückwärtszähler wird von externen Signalen getriggert, der Vorteiler bleibt unbenützt.
5 (nur bei Betriebsart „Zeitgeber")	0	Systemtakt wird vom Vorteiler um den Faktor 16 heruntergeteilt.
	1	Systemtakt wird vom Vorteiler um den Faktor 256 heruntergeteilt.
4 (nur bei Betriebsart „Zeitgeber")	0	Zeitmessung wird von negativer Flanke gestartet
	1	Zeitmessung wird von positiver Flanke gestartet
3 (nur bei Betriebsart „Zeitgeber")	0	Hat bit 2 den Wert Null, beginnt der Zeitgeber die Zeitmessung am Anfang des nächsten Maschinenzyklus. Hat bit 2 den Wert Eins, beginnt der Zeitgeber die Zeitmessung am Anfang des Maschinenzyklus, der auf das Laden der Zeitkonstante folgt.
	1	Hat bit 2 den Wert Null, beginnt der Zeitgeber die Zeitmessung nach dem laufenden Maschinenzyklus, sobald am Triggereingang die vorgesehene Flanke auftritt. Hat bit 2 den Wert Eins, beginnt der Zeitgeber die Zeitmessung nach dem Laden der Zeitkonstante, sobald am Triggereingang die vorgegebene Flanke auftritt.
2	0	Auf das Kanal-Steuerwort folgt keine Zeitkonstante. Die Zeitkonstante ist zum Anlaufen lassen des Zeitmeßvorgangs noch einzugeben.
	1	Das nächste Kontrollwort für den betreffenden Kanal stellt die Zeitkonstante für den Rückwärtszähler dar. Wird eine Zeitkonstante während eines Zeitmeßvorgangs eingegeben, wird zuerst die laufende Messung zuende geführt, bevor die neue Zeitkonstante in den Rückwärtszähler geschrieben wird.
1	0	Kanal zählt weiter
	1	Kanal unterbricht das Zählen, bis eine Zeitkonstante eingegeben wird. Ausgang ZC/TO ist inaktiviert, und der Kanal kann keinen Interrupt anfordern

Nach einem Steuerwort mit Bit 2=1 wird das nächstfolgende Steuerwort für den betreffenden Kanal als Zeitkonstante interpretiert und ins Zeitkonstantenregister des Kanals geladen.
Eine Steuerwortkonstante = 0 wird als Zeitkonstante = 256 interpretiert.

D7	D6	D5	D4	D3	D2	D1	D0
ZK7	ZK6	ZK5	ZK4	ZK3	ZK2	ZK1	ZK0

b) dem Steuerwort zur Eingabe des Interrupt-Vektors das vom Z80-CTC durch eine "0" im niederwertigsten Bit erkannt wird.

Während des Interrupt-Quittungs-Zyklus ("Interrupt-Acknowledge Cycle") wird dieser Vektor vom anfordernden Kanal höchster Priorität auf den Datenbus gelegt; vorher muß der Vektor dem CTC über Kanal Nr. 0 und Datenbit D0 gleich Null vorgegeben werden.
D 7 . . . D 3 enthalten dabei den Vektor; D 2 und D 1 sind beim Vektorladen unbenützt. Sobald der CTC auf eine Interrupt-Quittung reagiert, enthalten D 2 und D 1 den Binärcode der Nummer des höchstpriorisierten anfordernden Kanals, und D0 steht auf Null, da die Interrupt-Bedienroutine immer auf einer geraden Adresse beginnt.
Kanal 0 ist hardwaremäßig der höchstpriorisierte Kanal.

Format des zugehörigen Steuerworts:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	X	X	0

x = beliebiger Binärwert

2.9.4 Die Aussteuerung von Datenbus - Puffern

In Mikrokomputersystemen höherer Komplexität (d.h. mit größerer Anzahl von Speicher- und Ein/Ausgabe-Bausteinen) muß man die CPU-Busleitungen puffern, vor allem, wenn die Funktionseinheiten auf mehrere Baugruppen verteilt sind.
Im Allgemeinen kommt man dann zu einer Konfiguration nach Bild 36.
Man sieht, daß hier - wie allgemein üblich - die Datenleitungen die auf der gleichen Baugruppe wie die CPU untergebrachten Speicher- und Ein/Ausgabe-Bausteine direkt mit dem Datenbus verbunden sind.
Da der Datenbus bidirektional ist, muß dann den Datenbus-Pufferbausteinen gesagt werden, in welcher Richtung die Übertragung stattfindet; Die geschieht über das Steuersignal CPUIN. Damit bei Verteilung der Gesamtschaltung über mehrere Karten - keine Konflikte zwischen den Puffern auf den verschiedenen Karten vorkommen, muß außerdem festgelegt werden, welcher Pufferbaustein überhaupt aktiv sein darf (die übrigen sind auszuschalten). Hierzu dient das Steuersignal $\overline{CS}$.

Wenn man sich nun anhand von Bild 36 vergegenwärtigt, daß jeder der Pufferbausteine als Empfänger (von der Bus-Seite her gesehen) oder Sender arbeiten kann, sieht man, daß folgende Fälle des Datenflusses auftreten können:

1. Der Datentransfer findet innerhalb der Zentralbaugruppe statt.
2. Der Datentransfer findet statt zwischen Zentraleinheit und einer der beiden Zusatzeinheiten -

Im ersten Fall ist die Behandlung einfach: hier müssen lediglich die Datenbuspuffer auf der Zentralbaugruppe abgeschaltet werden, damit der "interne" Datentransfer nicht gestört wird. Im zweiten Fall sind folgende Unterscheidungen zu treffen:

2.1 Datentransfer zu Speicherbausteinen auf einer anderen Baugruppe

2.2 Datentransfer zu Speicherbausteinen auf einer anderen Baugruppe

2.3 Datentransfer von Speicherbausteinen auf einer anderen Baugruppe zur CPU

2.4 Datentransfer von Ein/Ausgabebausteinen auf einer anderen Baugruppe zur CPU

2.5 Interrupt-Behandlung

In den Fällen 2.1 bis 2.4 müssen die Datenbuspuffer auf jeder beteiligten Baugruppe aktiviert und ihre Datenübertragungsrichtung entsprechend der Transferrichtung festgelegt werden.

Sind in einem System, wie dem anfangs geschilderten, mehrere Baugruppen, die durch Datenbuspuffer voneinander getrennt sind, berechtigt, einen Interrupt anzufordern, so muß der Datenfluß vom anfordernden Baustein zur CPU gewährleistet sein. Fordert ein auf der Zentralbaugruppe befindlicher Ein/Ausgabe-Baustein einen Interrupt an, so muß der Datenbuspuffer auf der Zentralplatine gesperrt sein, damit der Interrupt-Vektor ungestört übertragen werden kann (das entspricht dem anfangs behandelten Fall 1). Fordert dagegen ein Ein/Ausgabe-Baustein auf einer anderen Baugruppe einen Interrupt an, so muß zum Zeitpunkt der INTA (Koinzidenz von $\overline{M1}$ und $\overline{IORQ}$) die Übertragung des Vektors von dort über den Datenbuspuffer der Zentralbaugruppe möglich sein.

Dazu ist notwendig:

a) nur die Datenbuspuffer zu aktivieren, die auf dem Weg zwischen der den Interrupt auslösenden Baugruppe und der CPU liegen

b) die Richtung dieser Puffer so festzulegen, daß die Datenübertragung zur CPU hin erfolgt.

Die hierbei notwendige Unterscheidung, welcher Ein/Ausgabe-Baustein einen Interrupt angefordert hat, ist durch dessen Interrupt Enable In- und Interrupt Enable Out-Leitung möglich:

Nur der Baustein, an dessen Interrupt Enable In-Leitung eine "logische" 1 und an dessen Interrupt-Leitung eine "logische" 0 liegt, kann einen Interrupt angefordert haben.

Aus dem obengenannten ist ersichtlich, daß die Verknüpfung mehrerer Steuerungssignale nötig ist, um die Datenbuspuffer richtig zu steuern.

Die Verknüpfung der einzelnen Signale läßt sich nach den folgenden Formeln mit Standard - Gatter - Bausteinen - oder besser mit einem einzigen 1k-bit-PROM-durchführen. Bitte beachten Sie, daß sich dabei durch die Festlegung der Übertragungsrichtung die im System erforderliche Bausteinstückzahl nicht erhöht, da die Freigabe der Pufferbausteine in jedem Fall eine Standard-Steuerlogik pro Baugruppe erfordert.

Die Verknüpfung ist nach folgenden Regeln vorzunehmen:

a) **Datenbuspuffer auf der CPU-Karte**

$$\text{CPUIN} = \underbrace{(\overline{\text{IMD}} \wedge \overline{\text{RD}})}_{\text{EXT MEM READ}} \vee \underbrace{(\overline{\text{IORQ}} \wedge \overline{\text{M1}})}_{\text{INTA}} \vee \underbrace{(\overline{\text{IORQ}} \wedge \overline{\text{IIODEC}} \wedge \overline{\text{RD}})}_{\text{EXT IO READ}}$$

$$\overline{\text{CS}} = \underbrace{(\overline{\text{IORQ}} \quad \overline{\text{M1}} \wedge \text{IIEI} \wedge \text{IIEO})}_{\text{EXT INTA}} \qquad \underbrace{(\overline{\text{IORQ}} \wedge \overline{\text{IIODEC}})}_{\text{EXT MEM TRANS}}$$

Dabei sind:

CPUIN	: Richtungssteuersignal	(High: Richtung zur CPU)
$\overline{\text{CS}}$	: Aktivierungssignal	(High: Puffer gesperrt)
$\overline{\text{M1}}$	: Maschinenzyklus 1	
$\overline{\text{IORQ}}$	: I/O Request	
$\overline{\text{IIODEC}}$	: ECB/C Internal I/O Anforderung	
$\overline{\text{RD}}$	: Read	
IIEO	: ECB/ C-Internal Interrupt Enable Out	
IIEI	: ECB/ C-InternalInterruptEnable In	
$\overline{\text{IMD}}$	: ECB/ C-Intern Speicheranforderung	

b) Datenbuspuffer auf weiterer I/O Karte

$$\overline{IOIN} = \underbrace{(\overline{IORQ} \wedge \overline{IODEC} \wedge \overline{RD})}_{\text{IO Write}} \vee \underbrace{(\overline{IORQ} \wedge \overline{M1} \wedge IEI \wedge IEO)}_{\text{INTA}}$$

$$\overline{CS} = 0$$

Dabei sind:

$\overline{IOIN}$	= Richtungssteuersignal	(High= zur betreffenden Karte hin)
$\overline{CS}$	= Aktivierungssignal	(High= Puffer gesperrt)
$\overline{M1}$	: Maschinenzyklus 1	
$\overline{IORQ}$	: I/O-Anforderung	
$\overline{IODEC}$	: I/O-Auswahl	
$\overline{RD}$	: Read	
IEI	: Interrupt Enable In	
IEO	: Interrupt Enable Out	

2.10 Anwendungsbeispiel für eine Parallel-Schnittstelle Thermodrucker

2.10.1 Vorbemerkung zum verwendeten Drucker

Der Kontron OEM-Drucker 5010 ist ein Streifendrucker auf Metallpapierbasis. Er kann auf 50mm breitem Papier bis zu 30 ASCII-Zeichen pro Zeile bei einer Geschwindigkeit von max. 2 Zeilen pro sec. drucken.
Er eignet sich durch seine geringen Abmessungen, die einfache Konstruktion und dem niedrigen Preis besonders gut zum Einsatz in Mikrocomputer-Anwendersystemen, in denen kleinere Datenmengen zu dokumentieren sind. (z.B. zur Protokollierung in Mikrocomputer-Steuer- und Regelsystemen oder zur Belegausgabe in Fertigungs-Überwachungsterminals).

2.10.2 Schaltung der Hardwareanpassung

Zum Anschluß der Druckers genügt ein halber Baustein Z80-PIO.
Der Druckvorgang beim 5010 läuft nach dem Zeitplan in Bild 50 ab.

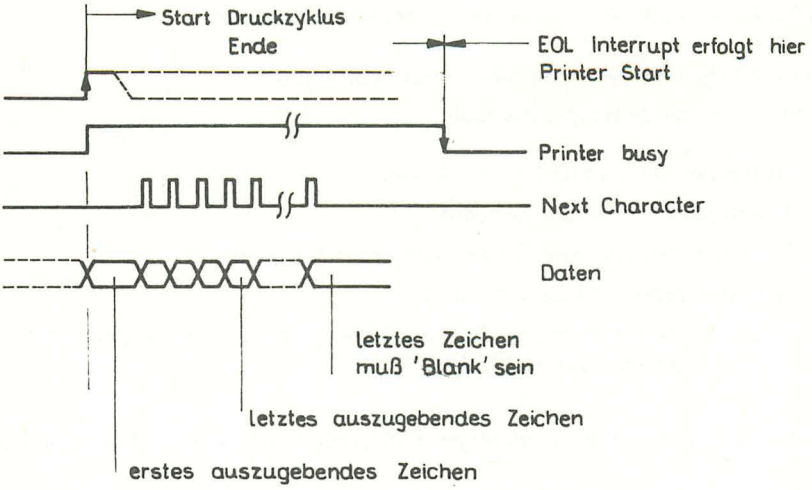


Bild 50: Drucker-Timing

Der Anschluß der einzelnen Druckerleitungen an den Mikrocomputer läßt sich dementsprechend in einfacher Form mit dem Z80-PIO nach dem in Bild 51 angegebenen Schema durchführen.

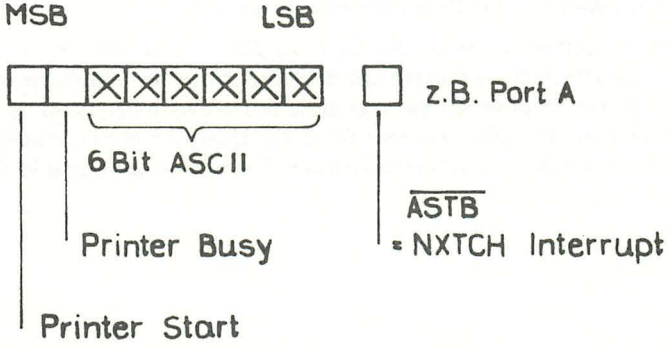


Bild 51: Verknüpfung PIO/Drucker

2.10.3 Zugehöriges Steuerprogramm ("Handling Routine")

Ein Flußdiagramm des Steuerprogramms zeigt Bild 52
Es enthält folgende Hauptaktivitäten:

a. Initialisierung des PIO (Output Mode)
b. Übertragung des ersten Zeichens
c. Warten auf das nächste Zeichen (Interrupt NXTCH);
d. wird wiederholt, bis Zeile zu Ende ist.

   Nach Ausgabe des letzten Zeichens Bereitschaft für "Neue Zeile" (Umschalten des Interrupts auf EOL-Signal)

Hierbei wird u.a. von zwei wichtigen Fähigkeiten des Systens Z80- Gebrauch gemacht.

a. Die "Betriebsart 3" der Z80-PIO (Control Mode), die die vom Anwenderprogramm kontrollierte Festlegung einzelner Leitungen eines Ports als Ein- oder Ausgabeleitungen erlaubt, wobei unter den Eingabeleitungen zusätzlich festgelegt werden kann, welche einen Interrupt auslösen sollen; in Verbindung mit der Möglichkeit, die Datenübertragungsart (Ein-, Ausgabe, bidirektional, Control) während des Programmablaufs umzudefinieren.
b. Die Interrupt-Architektur des Systems Z80, die es erlaubt, ein und dasselbe Port von verschiedenen Interrupt-Bedienroutinen behandeln zu lassen.
   Die dadurch in unserem Beispiel erzielte Hardware-Ersparnis ist evident; man kommt ohne ein oder mehrere Ports zur Übergabe der Drucker-Steuersignale aus. Das komplette Listing der Drucker-Steuerroutine zeigt Bild 52.

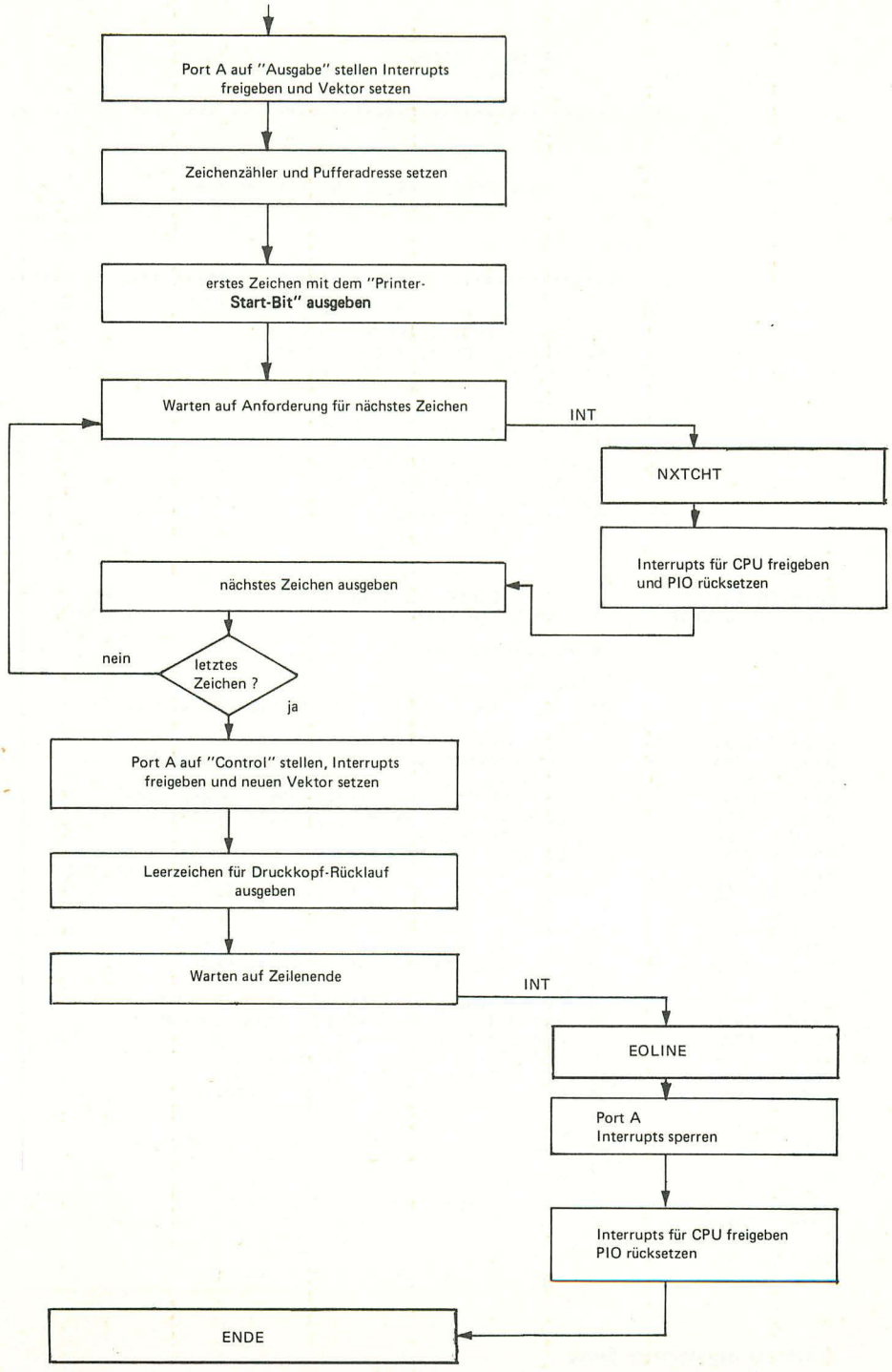


Bild 52: Programmablauf des Druckersteuer-Programms

```
                                OEM10  LISTING                    PAGE   1
LOC   OBJ CODE   STMT SOURCE STATEMENT                            ASM 3 0

                  1 ;*****************************************************
                  2 ;*
                  3 ;*
                  4 ;*
                  5 ;*       INTERFACE STREIFENDRUCKER 5010 =
                  6 ;*                                 ECB/C
                  7 ;*
                  8 ;*
                  9 ;*
                 10 ;*****************************************************
                 11 ;
                 12 ;
                 13 ;SOFWARE DRIVER ZUM ANSCHLUSS EINES
                 14 ;STREIFENDRUCKERS KONTRON-5010/5110
                 15 ;AN Z-80 PIO IN ZILOG ECB/C ODER MCB
                 16 ;
                 17 ;
                 18 ;BENOETIGT WIRD EIN PORT EINES PIO
                 19 ;
                 20 ;AN DAS PROGRAMM SCHLIESST SICH EIN
                 21 ;DATEN UEBERGABE BEISPIEL ZUM
                 22 ;AUSDRUCK DES TEXTES:
                 23 ; "ZILOG Z-80 MICRO" AN
                 24 ;
                 25 ;

PRINTER OUTPUT                  OEM10  LISTING                    PAGE   2
 LOC   OBJ CODE   STMT SOURCE STATEMENT                            ASM 3.0

                    26 *H PRINTER OUTPUT
                    27 ;
                    28 PRNTOUT:
                    29 ;    >>>>>>>>>>>>INITIALIZE PIO<<<<<<<<<<<<<<<<
                    30 ;
0000   3E0F         31  LD A,0FH
0002   D30A         32  OUT (PRNTCT),A    ;SET OPERATING MODE TO OUT
0004   3E87         33  LD A,87H
0006   D30A         34  OUT (PRNTCT),A    ;INTERRUPT CONTROL WORD
0008   3E5C         35  LD A,NXTC.AND.0FFH ;INTERRUPT VECTOR
000A   D30A         36  OUT (PRNTCT),A
000C   3E00         37  LD A,0
000E   ED47         38  LD I,A            ;HI BYTE OF INTERRUPT VECTOR
0010   ED5E         39  IM 2              ;INTERRUPT MODE 2
                    40 ;
                    41 ;
0012   0610         42  LD   B,10H        ;SET DIGIT COUNTER
0014   215B00       43  LD   HL,DATA+0FH  ;SET DATA ADDRESS
                    44 ;
0017   7E           45  LD   A,(HL)       ;FETCH 1ST CHARACTER
0018   E63F         46  AND 3FH           ;RESET PRINT COMMAND
001A   D308         47  OUT (PRNT),A      ;AND
001C   CBFF         48  SET  7,A          ; SET PRINT COMMAND BIT
001E   D308         49  OUT  (PRNT),A   ; AND 1ST CHARACTER
0020   FB           50  EI                ;ENABLE INTERRUPT
                    51 ;
                    52 HALT:
0021   76           53  HALT              ;WAIT FOR NEXT CHARACTER
                    54 ;
0022   2B           55  DEC  HL           ;FETCH
0023   7E           56  LD   A,(HL)       ; NEXT CHARACTER
0024   CBF7         57  SET 6,A           ;DONT PULL DOWN THE PRNTBUSY
                    58                    ;OUTPUT OF PRINTER
0026   D308         59  OUT  (PRNT),A   ; AND OUTPUT
0028   10CF         60  DJNZ HALT-$       ;COUNT CHARACTERS
                    61 ;
```

Fortsetzung nächste Seite

```
PRINTER END OF LINE          OEM10  LISTING                    PAGE   3
 LOC   OBJ CODE    STMT SOURCE STATEMENT                        ASM 3.0

                    62 *H PRINTER END OF LINE
                    63 ;
                    64 ;LAST CHARACTER IS OUT
                    65 ;PREPARE FOR END OF LINE (EOL):
                    66 ;
002A   3ECF         67  LD A,0CFH
002C   D30A         68  OUT (PRNTCT),A  ;SET PRINTER PORT TO CONTROL
002E   3E40         69  LD A,40H
0030   D30A         70  OUT (PRNTCT),A  ;DEFINE INPUT LINES
0032   3E97         71  LD A,97H
0034   D30A         72  OUT (PRNTCT),A   ;INTERRUPT CONTROL WORD
0036   3EBF         73  LD A,0BFH
0038   D30A         74  OUT (PRNTCT),A   ;INTERRUPT MASK
                    75 ;
                    76 ;
                    77 ;
003A   3E20         78  LD   A,' '        ;HOLD "BLANK"
003C   D308         79  OUT  (PRNT),A  ; IN PORT
                    80 ;
003E   76           81  HALT             ;WAIT FOR END OF LINE
                    82 ;
                    83 ;LINE READY
                    84 ;
003F   76           85  HALT
0040   76           86  HALT
0041   76           87  HALT
                    88 ;


INT  SERVICE AND DATA        OEM10  LISTING                    PAGE   4
 LOC   OBJ CODE    STMT SOURCE STATEMENT                        ASM 3.0

                    89 *H INT. SERVICE AND DATA
                    90 ;
                    91 ;
                    92 NXTCHT:            ;"NEXT CHARACTER" INT.SERVICE
0042   FB           93  EI                ;ENABLE CPU FOR INTERRUPTS
0043   ED4D         94  RETI              ;RETURN FROM INTERRUPT
                    95 ;
                    96 EOLINE:            ;"END OF LINE" INT.SERVICE
0045   3E03         97  LD A,3
0047   D30A         98  OUT (PRNTCT),A   ;RESET INT ENABLE FF IN PIO
                    99 ;
0049   FB          100  EI                ;ENABLE CPU FOR INTERRUPTS
004A   ED4D        101  RETI              ;RETURN FROM INTERRUPT
                   102 ;
                   103 DATA:              ;DATA TO BE PRINTED:
                   104 ;
004C   5A494C4F    105  DEFM 'ZILOG Z-80 MICRO'
                   106 ;
005C               107  DEFS $.MOD.2      ;INT. SERVICE TABLE ON ODD
                   108                    ;ADDRESS
                   109 ;   INTERRUPT SERVICE TABLE
                   110 ;
005C   4200        111 NXTC:  DEFW NXTCHT ;NEXT CHARACTER
005E   4500        112 EOL:   DEFW EOLINE ;END OF LINE
                   113 ;
                   114 ;
                   115 PRNT EQU 8         ;PIO ADD. ON ECB/C
                   116                    ;PORT A
                   117 PRNTCT EQU 0AH     ;CONTROL PORT
                   118 ;
                   119  END

   0 ASSEMBLY ERRORS
```

Die Länge der Routine ist ca. 70 Byte.

Bild 53: Listing des Druckersteuer-Programms

2.11. Beeinflußung des Programmablaufs über Schaltkontakte

Der Anwender sieht sich häufig mit folgenden Problemen konfrontiert:

a) Es gibt n Schalter (Einfach-Kontakte), deren Betätigung jeweils einen Interrupt bzw. den Sprung zu einer Interrupt-Service-Routine auslöst.

b) All diese Schalter sind in vorgegebener Weise zu priorisieren.

c) Jedem Schalter ist in eindeutiger Weise eine Interrupt-Service-Routine zugeordnet, deren Anfangsadresse an einer beliebigen Stelle im max. 64KByte großen Programmspeicher steht.

Dieses Problem ist mit 1 Baustein Z80-CTC pro 4 Schalter/Interrupt-Service-Routinen ohne weitere Logik zu lösen. Jeder Eingang CLK/TRG ist mit einem der Schalter verbunden (Low- oder High-Aktiv). Die Entprellung des Schalters ist hardwaremäßig über ein Flip-Flop oder softwaremäßig über eine entsprechende Warteschleife nach Bearbeitung des Interrupts durchzuführen.

Jedem der 4 Kanäle ist eine Speicheradresse zugeordnet, in der jeweils die Anfangsadresse der zum Interrupt zugehörigen Service-Routine steht. Die erste dieser 4 Adressen wird mit einem Steuerwort (letztes bit = 1) gewöhnlich zu Programmbeginn von den CPU in den CTC geladen.
Die 4 Kanäle eines jeden CTC sind untereinander hardwaremäßig priorisiert. Eine Erweiterung auf mehr als 4 Kanäle ist durch Kaskadierung mehrerer Bausteine über deren 'Daisy Chain" (IEI/IEO) möglich.
Eine Änderung dieser Priorisierung ist jederzeit durch die Möglichkeit gegeben, jeden der Bausteine individuell zu "disablen" d.h. ihre Interrupt Meldung zu sperren, wobei trotzdem die Interrupt-Meldung der Peripherie nicht verloren geht, sondern solange gespeichert bleibt, bis sie durch die CPU bearbeitet wird.
Es muß selbstverständlich sichergestellt sein (beispielsweise durch Quittungslogik), daß inzwischen keine neue Interruptanforderung erfolgt.
So würde beispielsweise der Kanal hardwaremäßig niedrigster Priorität die höchste Priorität bekommen, indem per Programm alle übrigen Kanäle vorrübergehend gesperrt werden.
Zu Programmbeginn wird jedem Kanal ein Steuerwort übermittelt, das ihn in Betriebsart "Ereigniszähler ' bringt und seinen Rückwärtszähler über das Zeitkonstantenregister auf den Wert 1 setzt (vorher ist ein Steuerwort mit bit 2=1 auszugeben).

Die Betätigung eines der Schalter löst nun ein Dekrementieren des Rückwärtszählers des CTC aus, der dadurch den Wert 0 erreicht und einen Interrupt auslöst, soweit er nicht softwaremäßig gesperrt ist. Die Programmausführung wird nun nach Abarbeitung des momentan in Ausführung befindlichen Befehls unterbrochen und bei der Anfangsadresse der Interrupt-Service-Routine fortgesetzt.
Diese ist gleich dem Inhalt der Arbeitsspeicheradresse, die sich aus Inhalt des CTC-Interrupt-Vektorregisters als niederwertiges Byte und dem Inhalt des Interrupt

(I -) Registers der CPU als höherwertiges Byte ergibt. Der Rücksprung ins Hintergrundprogramm erfolgt in üblicher Weise mit einer RETI-Anweisung. Vorher muß jedoch dafür gesorgt werden, daß das CTC-Zeitkonstantenregister wieder auf "1" gesetzt wird.

Falls eine Priorisierung der einzelnen interrupt-anfordernden Schalter bereits in der Peripherie mit einer Daisy-chain-ähnlichen oder sonstigen Hardware durchgeführt wurde, kann die Auslösung der Interrupts über einen oder mehrere PIO-Bausteine erfolgen.
Die betreffenden PIO-Ports arbeiten dabei in der Betriebsart "Bit-Ein/Ausgabe" ("=Control Mode"), wobei über Steuerworte die Möglichkeit der Interruptanforderung jede einzelne Leitung freigegeben oder gesperrt werden kann; darüberhinaus sind Bedingungen zur Erzeugung von Interrupts aufgrund logischer Verknüpfungen programmierbar.

Der Vorteil dieses Verfahrens gegenüber der Verwendung eines CTC ist, daß mit einem einzigen PIO-Baustein bis zu 16 Interruptbedingungen verknüpft, maskiert und geprüft werden können, nachteilig kann hierbei sein, daß man den einzelnen Schaltern die Einsprungadressen der Bedienroutinen über ein Verzweigungsprogramm zuordnen muß, da ja hardwaremäßig pro 8 bit (= 1 PIO-Kanal) zunächst nur eine einzige Bedien-Routinenadresse zur Verfügung steht.

3. Betriebsbereite Mikrocomputer-Systeme

In den bisherigen Abschnitten haben wir schon eine ganze Reihe von Merkmalen besprochen, die die Mikrocomputertechnik von der "klassischen" Rechnertechnik unterscheidet und dadurch weitere Möglichkeiten eröffnen, Computertechnik für neuartige Zwecke auszunützen.
Ein weiterer Schritt in diese Richtung ist die Verfügbarkeit kompletter Mikrocomputersysteme die nach Anschalten der Versorgungsspannung an die gewünschte Peripherie angeschlossen werden können und dann sofort betriebsbereit sind und dadurch den Aufwand zur Entwicklung eines rechnergesteuerten Systems auf ein Minimum reduzieren.

Wie wir bereits gesehen haben, ist für solche "General Purpose"-Zwecke natürlich neben der Hardware (also dem Rechner selbst) auch eine Betriebssoftware nötig, die mit dem Mikrocomputersystem mitgeliefert wird.

Es sollen hier drei Systeme vorgestellt werden:

* Z80-MCS, das voll ausbaufähig ist und Massenspeicher beinhaltet und damit für komplexere Anwendungen prädestiniert ist
* Z80-PDS, eine Billigausführung des MCS mit nur 1 Floppy-Disk und
* Z80-KIT, das bereits Tastatur und Hexadezimalanzeige enthält und damit für Billigst-Anwendungen geeignet ist, da hier die Erfordernis nach aufweniger Bedienungsperipherie entfällt.

3.1. OEM-Mikrocomputer-System mit eingebauten Massenspeicher

Das Z80-MCS Mikrocomputersystem ist ein Allzweckcomputer mit hoher Leistungsfähigkeit zu einem relativ niedrigen Preis. Er eignet sich insbesonders zur Realisierung von Mikrocomputeranwendersystemen, die Floppy Disks als Massenspeicher verwenden, ohne jeglichen Hardware-Entwicklungsaufwand und mit einem stark reduzierten Software-Entwicklungsaufwand. Durch die hohe Integration der Elektronik- und den ausgereiften Stand der Technik der verwendeten Floppy-Disks Laufwerke wurde eine hohe Zuverlässigkeit des Systems erreicht.
Das MCS beinhaltet in seiner Grundkonfiguration neben der Zentraleinheit 16 kByte Schreib/Lese-Speicher, 3 kByte Festwertspeicher, Parallel- und Serienschnittstellen und 2 Floppy Disk Laufwerke. Die Speicherkapazität des vom Z80-CPU adressierbaren Schreib/Lese-Speichers ist bis zu 64 kByte ausbaubar.

3.1.1. Systemkonzept

Das Zilog Mikrocomputersystem Z80-MCS ist ein weiterer Schritt in Richtung der Verwendung vorhandener Hardware- und Software-Standards auf dem Gerätesektor und Datenverarbeitungssektor anstelle von jedesmal neu zu entwickelnden Speziallösungen. Das Z80-MCS ist so konzipiert, daß es direkt als OEM-Einheit in einem Mikrocompuer-/Anwendersystem zu verwenden ist. Der Entwicklungsaufwand beschränkt sich auf das Design der Interfaceschaltungen und des Austestens eines Anwenderprogramms, wobei auch dabei selbstverständlich auf Anwenderprogramme des Z80- und der 8080A-Programmbibliothek zurückgegriffen werden kann. Darüberhinaus ist auf dem MCS die gesamte Z80-Betriebssoftware ablauffähig, wie der absolute Assembler, der relokative Assembler (RIO), Basic Interpreter, PL/Z-Compiler, Cobol-Compiler, Fortran-Compiler usw.

3.1.2. Beschreibung der Anlage

Das Z80-MCS ist eine nach Anlegen der Versorgungsspannung (220 V, 50 Hz) sofort arbeitsfähige Anlage mit 2 Floppy-Disk-Laufwerken. Der Dialog mit dem Computersystem findet über eine Serienschnittstelle (RS 232 oder 20 mA Stromschleife) statt, wodurch der Anschluß eines jeden Standardbildschirms oder Teletypeähnlichen Fernschreibers möglich ist. Die Datenübertragungsfrequenz ist im Bereich von 110 bis 38.600 baud einstellbar. Zusätzlich zu diesem Anschluß der Bedienkonsole sind an der Rückwand des Systems 7 weitere Plätze für Stecker zum Anschluß von Prozeß- oder sonstiger Peripherie vorgesehen. Die freien Kartensteckplätze können durch Wire-wrap-Technik vom Anwender entsprechend der von ihm verwendeten Karten beliebig verdrahtet werden.

Insgesamt sind im Chassis des MCS neun Steckplätze verfügbar, von denen zwei in der Minimalsystemkonfiguration (Computerkarte und Floppy-Disk-Steuerungskarte) besetzt werden. In die übrigen Steckplätze lassen sich die optional angebotenen Karten für Speicher und Ein/Ausgabeerweiterungen oder aber beliebige kundenspezifische Baugruppen einstecken. Der Kontakt der anzuschließenden Peripherie

erfolgt in all diesen Fällen über die auf der Rückseite des Systems vorgesehenen Ein/Ausgabstecker. Vom System können bis zu acht Floppy-Disk-Laufwerke ohne externe Decodierung adressiert werden.

Auf der Computerkarte stehen vier Sockel für Festwertspeicher zur Verfügung, von denen drei durch die standardmäßig mitgelieferte Betriebsfirmware belegt sind. Der Benutzer kann entweder die verbleibende Festwertspeicherfassung für seine spezifischen Zwecke verwenden oder aber auch die eingesteckten Festwertspeicher herausnehmen, wodurch er dann alle vier Sockel zur seiner freien Verfügung hat. Auch parallele Ein/Ausgabe ist bereits auf der Rechnerbaugruppe des MCS aufgebaut; auch stehen vier 16 Pin Sockel zum Einsetzen anwendungspezifischer Treiberbausteine nach dem Parallel-Ein-Ausgabe-Baustein zur Verfügung. Um den Anwender in den Möglichkeiten des Systemdesigns nicht einzuschränken, wurden diese Fassungen nicht mit den Anschlüssen des PIO's verdrahtet; auf diese Weise kann die Anschlußbelegung vom Anwender durch Legen von Drahtbrücken festgelegt werden.
Für Echtzeitaufgaben wurde ein Z80-CTC-Baustein eingesetzt, der die Festlegung der Übertragungsrate der Bedienkonsole und die Floppy-Disk-Synchronisation vornimmt. Zwei seiner Kanäle stehen dem Benutzer zur Realisierung von Echtzeituhr oder ähnlichen Anwendungen zur freien Verfügung. Der in dem System enthaltene Ein/Ausgabe-Port-Adreßdecoder adressiert einen Adreßbereich von 32 zusammenhängenden Ein/Ausgabeadressen. Einige davon werden für interne Zwecke verwendet; es besteht volle Erweiterbarkeit auf den von der Z80-CPU adressierbaren Ein/Ausgabebereich. Durch Verlegung von Drahtbrücken kann man auf der Zentralkarte Speicher und Ein/Ausgabebereiche nach Wunsch festlegen. Im übrigen ist die Erweiterbarkeit durch volle Pufferung sämtlicher Systembussignale gesichert. Ein Rücksetzschalter ist an der Vorderseite des Gehäuses angebracht, der Netzschalter befindet sich auf der Rückseite. Zum Standardlieferumfang des Systems gehört ein vollständiges Betriebssystem mit Dateiverwaltung, Editor, Makro-Assembler und Fehlersuchprogrammen.

3.1.3. Erweiterungsmöglichkeiten

Die Pufferung der Zentralbaugruppe im MCS und die sieben freien Kartensteckplätze erlauben einfach durch Einstecken von Standard- oder anwendungsspezifischen Baugruppen eine Erweiterung der Fähigkeiten des MCS.

Eine Erweiterung ist um folgende Baugruppen möglich:

* Z80-RMB: Schreib/Lesespeicher-Erweiterung mit 16 kByte Kapazität, aufgebaut mit 4 kByte dynamischen RAM-Speicherbausteinen. Zusätzlich auf dieser Baugruppe untergebracht sind acht Fassungen für Festwertspeicherbausteine.

* Z80- IOB: Parallel-Ein/Ausgabe-Erweiterung zur Erweiterung des MCS-Systems um acht Ein/Ausgabe-Ports à 8 bit incl. Steuerleitungen.

* Z80-SIB: Serielle Ein/Ausgabe Erweiterung des Z80-MCS um vier serielle Ein/Ausgabe Duplexschnittstellen.

* Z80-VDB: Ermöglicht den direkten Anschluß des Z80-MCS an einen beliebigen Fernsehbildschirm (Heim- oder Industriemonitor). Mit dem Z80-VDB ist sowohl alphanumerisches als auch graphisches Arbeiten möglich. In der alphanumerischen Betriebsweise sind ca. 2 kByte, in der graphischen Betriebsweise sind 20 kByte Bildwiederholspeicher nötig. Über einen Z80-PIO-Baustein auf der Baugruppe ist der Anschluß einer Standard-ASCII-Tastatur möglich.

* Z80-PMB: Festwertspeicher-Erweiterungsbaugruppe mit Parallel-Ein/Ausgabe-Erweiterung . Auf der Baugruppe untergebracht sind 16 Fassungen für Festwertspeicher-Bausteine und 2 Parallel-Ein/Ausgabe-Bausteine (Z80-PIO).

* Z80-CPB: PROM Programmierbaugruppe. Diese Baugruppe wird mit der zugehörigen Betriebssoftware geliefert und erlaubt das Programmieren von bipolaren und MOS-Festwertspeicher-Bausteinen (PROM's und EPROM's). Folgende Festwertspeichertypen lassen sich elektrisch direkt ohne Eingriff in der Schaltung des Z80-CPB programmieren: 2704, 2708, Harris 7620, 7621,7640 und 7641.

* Analog/Digital- und Digital/Analog-Wandlerkarten.

Insbesondere ist hier die Erweiterbarkeit und die Baugruppe Z80-VDB bemerkenswert.
Mit ihr erspart man sich teure Bedienkonsolen (wie Datensichtgeräte, Fernschreiber usw.) und kann sogar graphische Darstellungen durchführen. Die über 160 000 dabei zur Verfügung stehenden Bildpunkte und der Befehlssatz der Z80-CPU, der mit einem einzigen Befehl Speicher-Such- und Transfer-Zugriffe bis zu 64 Byte erlaubt, fordern geradezu zur Bildverarbeitung oder wenigstens zur graphischen Darstellung von Vorgängen und Prozeßparametern heraus.

Außerdem erlaubt diese Baugruppe den Anschluß einer preisgünstigen Standard-ASCII-Tastatur, wodurch die Serienschnittstelle des Gerätes für Datenfernübertragung über das Telefonnetz frei wird.
Man sieht, daß die Anwendungsmöglichkeiten dieser Anlage praktisch nur noch durch die Phantasie des Anwenders begrenzt wird.

3.1.4 Das System Z80-PDS

entspricht dem bisher beschriebenen Z80-MCS, jedoch beinhaltet es nur 1 Floppy-Disk-Laufwerk und erlaubt eine Maximalerweiterung um 2 Baugruppen der MCB-Serie.

3.2. Mikrocomputersystem zur Lösung einfacher und kostenkritischer Probleme

Nach unserem Exkurs in die Welt der komplexen Systeme wollen wir Ihnen nun eine Möglichkeit vorstellen, den Mikrocomputer bei Anwenderproblemen einzuführen, die nur minimale Hardwarekosten erlauben.

In solchen Fällen ist es nötig,

- komplexe Mikrocomputer-Peripherie Bausteine einzusparen und
- aufwendige Bedienungsperipherie durch einfache Tastenfelder und Hexadezimalanzeigen zu ersetzen.

Ein derartiges Gerät ist der Z80-KIT; er läßt sich durch Verwendung eines entsprechenden Betriebsprogramms zur Entwicklung von Anwenderprogrammen auf Hexadezimalebene verwenden, aber selbstverständlich auch mit einem entsprechenden Anwenderprogramm zusammen mit der gewünschten Prozeßperipherie als Anwender-Computersystem verwenden.

Basis dieses Mikrocomputers ist die Grundplatine, auf der die Bedienungstastatur die Anzeigeeinheiten und die dazu notwendige Steuerelektronik untergebracht sind. Ebenfalls auf dieser Grundplatine wurde ein System-Bus realisiert, der sämtliche Adreß-, Daten und Steuersignale über fünf Vielfachsteckverbindungen zur Verfügung stellt. Über einen der genannten Steckplätze erfolgt der Anschluß eines Einplatincomputers. Erweiterungsmöglichkeiten bezüglich Speicher etc. sind durch die restlichen Bus-Anschlüsse gegeben.

3.2.1. System-Eigenschaften

Die serienmäßige Grundausstattung umfaßt folgende Systemkomponenten:

Grundplatine

*	Eingabetastatur:	26 Tasten zur Bedienung des KIT
*	Anzeigeeinheit	4 hexadezimale Anzeigen für Adressen und 2 hexadezimale Anzeigen für Daten

Einplatinencomputer

*	Mikroprozessor	1 x Z80-CPU/PS
*	Taktfrequenz	2,5 MHz
*	Programmspeicher	Z80-KIT-Betriebssoftware (= 1 kByte)
*	Schreib/Lesespeicher	2 x 2112/A-4 Statische RAM's (= 1/4 kByte)
*	Ein/Ausgabeeinheiten	1 x Z80-PIO 2 x 8 bit Programmierbare Parallel-Schnittstelle 1 Serienschnittstelle für Audio-Cassette oder TTY

3.2.2. Erweiterungsmöglichkeiten

Ohne Veränderung des bestehenden KIT ist eine Erweiterung durch Zusatzplatinen möglich. Angeboten werden folgende Erweiterungspakete:

- Z80-KIT/PPI — Programmier- und Peripherie-Interface (PROM-Programmierbaugruppe)
- sämtliche Baugruppen der Z80-ECB-Serie:
  * ECB/E (8kPROM + 1k stat. RAM)
  * ECB/I (4 Achtbit-Parallel-Ein/Ausgabe-Ports + 4 Zähler/Zeitgeber-Kanäle)
  * ECB/V (4 kByte CMOS-RAM, nicht-flüchtig, mit Speicherschutzschalter)
  - Analog/Digital und Digital/Analog-Baugruppen

Der Z80-KIT läßt sich aufgrund seiner Architektur und seines elektronischen Aufbaus auf insgesamt fünf Platinen erweitern, wobei die vier Zusatzplatinen nach Wunsch gewählt werden können.

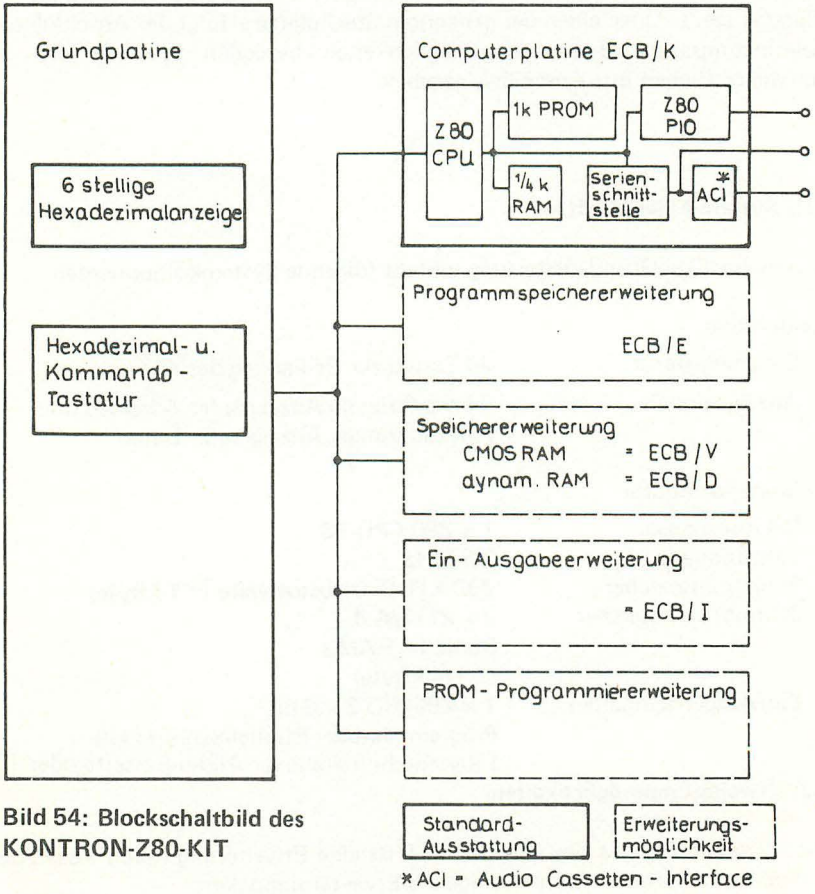


Bild 54: Blockschaltbild des KONTRON-Z80-KIT

3.2.3. Die Bedienungs- und Anzeigeeinheit

Im Sinne minimaler Hardwarekosten wurde von zwei Voraussetzungen ausgegangen:

- Verwendung billiger Standard-Drucktasten mit einem einzigen Arbeitskontakt
- Vermeidung des Einsatzes eigener Ein/Ausgabe-Bausteine und Mitbenützung der Adreßleitungen zu Ein-/Ausgabezwecken.
  Man definiert 2 Tasten-Puffer von je 4 Wörtern: Tastenneu-Puffer TNP und Tastenalt-Puffer TAP. Am Anfang werden alle Puffer gelöscht. Danach wird eine Programmschleife dadurch eingeleitet, daß die nicht mehr aktuelle TNP in den TAP transferiert wird.
  Anschließend ist TNP zu reaktualisieren.

Man untersucht jetzt die beiden Tasten-Puffer auf "Neuzugänge", indem man gleiche Bitpositionen in beiden Puffern miteinander vergleicht. Damit die Tasten voneinander programmtechnisch leicht unterscheidbar sind, ordnet man jeder Taste einen eigenen Tastencode zu. Dieser setzt sich aus einer Zeilen- und einer Spaltenklemmung zusammen. Vorausgesetzt, daß nicht mehrere Tasten gleichzeitig gedruckt sind, sucht die folgende Routine die "Neuzugänge", ermittelt den Tastencode und legt ihn im "Tastenregister" ab.

Wiederholt man diese Routine zyklisch für immer neue TNP, so gewinnt man in dem Tastenregister alle Tastencodes in der Reihenfolge, in der die Tasten gedrückt worden waren. Nun können die eingegebenen Werte (= Tastencodes) weiterverarbeitet werden.

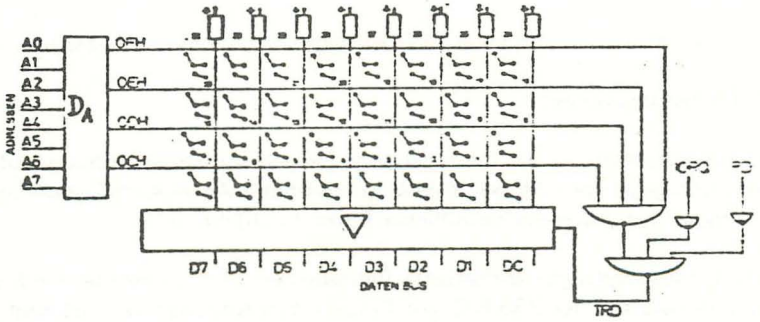


Bild 55: Direkter Anschluß eines Tastenfeldes an einen Mikroprozessor

Bei der Konzeption der Low-Cost-Anzeige wurde von den gleichen Voraussetzungen ausgegangen. Die Anzeige umfaßt 6 ungepufferte Standard-7-Segment-LED's, die ohne Zwischenschaltung eines Hexadezimal/7-Segment-Decoders direkt per Tabelle aus dem Anwendungsprogramm angesteuert werden.

Dadurch können beliebige numerische, mit gewissen Restriktionen auch hexadezimale und alphanumerische Anzeigen durch Vorgabe einer geeigneten Tabelle im Anwenderprogramm ausgegeben werden.
Den anzuzeigenden 6stelligen Zahlenwert speichert man zunächst in binärer Form. Die Umwandlung auf 7-Segment-Darstellung erfolgt über den Zugriff auf die 7-Segment-Tabelle SEGM, die zu jeder Zeile das entsprechende 7-Segment-Äquivalent enthält. Die Port-Adressen werden über die "PORT-Liste" angesprochen.

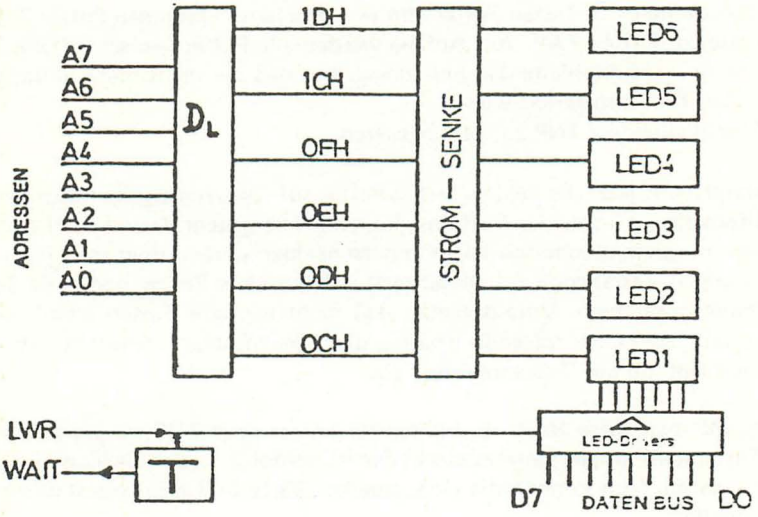


Bild 56: Direkter Anschluß einer LED-Anzeige an einen Mikroprozessor

3.2.4. Bedienungskonzept

Die Computerschaltung und die zugehörige Betriebssoftware wurden so aufgebaut, daß mit den gegebenen Mitteln eine Mikrocomputer-Anwenderprogrammentwicklung in möglichst anwenderfreundlicher Form durchführbar ist.

Soweit das festwertspeicher-residente Betriebsprogramm verwendet wird, sind die 13 Funktionstasten des Z80-KIT zur Eingabe von Kommandos bestimmt, die die einzelnen Betriebssystemfunktionen auslösen, und die verbleibenden 16 Hexadezimal-Tasten zur Eingabe von Daten, Adressen und sonstigen Parametern.
Benutzt man diese Betriebssoftware nicht oder nur teilweise, so stehen alle Tasten dem Anwender zur freien Verfügung; er kann ihre Bedeutung durch ein selbsterstelltes Anwenderprogramm nach Belieben festlegen.

RES — Die Rücksetzung (RESET) Taste gibt die Möglichkeit, das System zu jeder Zeit in einen definierten Anfangszustand zu bringen. Sie ist nach jedem Einschalten der Speisespannung zu betätigen.

EX — Ausführungskommando
veranlaßt das Mikrocomputersystem zur Ausführung der zuvor eingetippten Anweisung.
Es läßt das Bedienprinzip deutlich werden, daß in der gesamten Computertechnik üblich ist:

* die Eingabe der gewünschten Anweisung und
* die Nachricht an den Computer, daß das "Eingetippte" auch wirklich vom Computer getan werden soll.

Durch diese Zweiteilung wird der Computer davor geschützt, falsche Aktivitäten auszuführen, die durch Tippfehler des Anwenders zustande kommen. Die Bedienperson hat stattdessen die Möglichkeit, versehentlich gegebene Anweisungen durch Drücken einer Rücknahme- ("Storno"-, s.dort)-Taste zu annullieren.

LOAD — Einlesen einer Datei vom Serien-Interface

STORE — Ausgabe einer Datei über das Serien-Interface

DIS — Anzeigenanweisung (DISPLAY)
dient zur Ausgabe eines Speicher- oder Registerinhalts

SET — Schreibanweisung
dient zur Eingabe eines Wertes in eine Speicherzelle bzw. in ein Register

MEM — Speicheranweisung (MEMORY)
wird benötigt, wenn eine Speicheroperation durchgeführt werden soll (z.B. SET MEM ... gibt an, daß eine Speicherzelle belegt werden soll)

START — Startkommando
dient zum Starten eines Anwenderprogramms

STEP — Einzelschrittkommando
bewirkt Ausführung eines Programmschritts (= CPU-Befehl)

BR — Haltepunktanweisung (BREAKPOINT)
mit dieser Taste kann ein Haltepunkt auf eine gewünschte Adresse gelegt werden

IN — Eingabekommando
versetzt den Mikrocomputer in den Eingabemodus für die Eingabe eines Anwenderprogramms

IDM — Ausgabekommando (INCREMENT AND DISPLAY MEMORY)
versetzt den Mikrocomputer in den Ausgabemodus für die Ausgabe zusammenhängender Speicherbereiche.

STORN — Stornokommando
dient zum Löschen falscher oder irrtümlich eingegebener Werte und Anweisungen. Kann gegeben werden **bevor** die EX Taste gedrückt wurde. Danach ist das **gesamte** Kommando erneut einzutasten.

0/'	Hexadezimalziffer 0 / Kennung der Zweitregister (z.B. H' L' etc.)
1	Hexadezimalziffer 1
2	Hexadezimalziffer 2
3/I	Hexadezimalziffer 3 / I Register
4/P	Hexadezimalziffer 4 / Befehlszähler (Programm Counter = PC-Registerpaar)
5/S	Hexadezimalziffer 5 / Stack Pointer (SP-Registerpaar)
6/Y	Hexadezimalziffer 6 / IY Registerpaar
7/X	Hexadezimalziffer 7 / IX Registerpaar
8/H	Hexadezimalziffer 8 / H Register
9/L	Hexadezimalziffer 9 / L Register
A	Hexadezimalziffer A / A Register
B	Hexadezimalziffer B / B Register
C	Hexadezimalziffer C / C Register
D	Hexadezimalziffer D / D Register
E	Hexadezimalziffer E / E Register
F	Hexadezimalziffer F / F Register

3.2.5. Kommandosprache

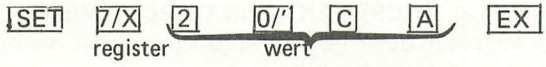
3.2.5.1. Setzen von Registern auf einen gewünschten Wert

Dies wird durch folgende Tastenfolge erreicht:

[SET] [register] [wert] [EX]

Für "register" ist die Taste zu drücken, die dem gewünschten Register zugeordnet ist. "wert" ist durch den gewünschten Wert in hexadezimaler Form zu ersetzen und zwar beginnend mit der höchsten Hexadezimalzahl.

Beispiel: Das 16 bit Register IX soll auf den Wert 20CAH gesetzt werden



Das 8 bit Register A' soll auf den Wert 07H gesetzt werden;

3.2.5.2. Anzeige von Registerinhalten (DISplay)

Dies wird durch folgende Tastenfolge erreicht:

DIS register EX

Für "register" ist wiederum die Taste zu drücken, die dem gewünschten Register zugeordnet ist. Der Inhalt des Registers erscheint an den Anzeigenelementen und zwar bei ADRESSE (4 Stellen), wenn es sich um ein 16 bit Register handelt (z.B. PC) bzw. bei DATEN (2 Stellen), wenn es sich um ein 8 bit Register handelt (z.B. A Register)

Beispiele: Der Inhalt des 8 bit Registers H soll angezeigt werden

DIS 8/H EX

Das Ergebnis erscheint bei DATEN (2stellig!)

Der Inhalt des 16 bit umfassenden Stack Pointers (SP-Register) soll angezeigt werden

DIS 5/S EX

Das Ergebnis erscheint bei ADRESSE (4stellig)

3.2.5.3. Eingeben eines Wertes in eine Speicherzelle

Hierfür sind zwei Kommandos notwendig

- Der Befehlszähler wird mit dem zuvor gezeigten Kommando auf die gewünschte Speicheradresse gesetzt. (Der Wert erscheint bei ADRESSE)

  SET 4/P adresse EX

- Anschließend wird die Tastenfolge

  SET MEM wert EX

  gegeben.

  Der Ausdruck "wert" ist durch den gewünschten Wert zu ersetzen.

Beispiele: Die Speicherzelle mit der Adresse 3C83 soll den Wert FF erhalten

SET 4/P 3/I C 8/H 3/I EX

Bei ADRESSE erscheint 3C83

SET MEM F F EX

Bei Daten erscheint FF

Die Speicherzelle mit der Adresse 3CC7 soll den Wert 12 erhalten

[SET] [4/P] [3/I] [C] [C] [7/X] [EX]

Bei Adresse erscheint 3CC7

[SET] [MEM] [1] [2] [EX]

Bei Daten erscheint 12

3.2.5.4. Ausgabe des Inhaltes einer Speicherzelle auf die Anzeige

Auch hier sind zwei Kommandos notwendig. Das erste Kommando dient dabei wie vorher zur Identifizierung der gewünschten Zelle, während das zweite Kommando die Ausgabe ihres Inhalts verursacht.

[SET] [4/P] [adresse] [EX]

[DIS] [MEM] [EX]

Beispiele: Der Inhalt der Speicherzelle 0083 soll ausgegeben werden

[SET] [4/P] [0/'] [0/'] [8/H] [3/I] [EX]

Bei Adresse erscheint 0083

[DIS] [MEM] [EX]

Bei DATEN erscheint der Inhalt dieser Speicherzelle

Der Inhalt der Speicherzelle 3CFF soll ausgegeben werden

[SET] [4/P] [3/I] [C] [F] [F] [EX]

Bei ADRESSE erscheint 3CFF

[DIS] [MEM] [EX]

Bei DATEN erscheint der Inhalt dieser Speicherzelle.

3.2.5.5. Eingabe eines Anwenderprogrammes

Für die Eingabe von Programmen ist der Eingabemodus vorgesehen, der das bequeme Eingeben auch längerer Programmteile ermöglicht. Das Setzen des Befehlszählers ist dabei nur einmal nötig und geschieht danach automatisch.

a) Als erstes wird der Befehlszähler auf die Adresse gesetzt, auf der der erste Befehl des Anwenderprogrammes stehen soll

[SET] [4/P] [adresse] [EX]

b) Daraufhin wird der Eingabemodus eingeleitet, indem die Taste

[INPUT]

gedrückt wird.

c) Nun kann man ein Programm eingeben, und zwar in folgender Weise

[X] [X] [EX]
[X] [X] [EX]
[X] [X] [EX]

gewünschte Daten in hexadezimalem Format

Der Befehlszähler wird bei jedem Schritt um 1 erhöht und zeigt auf die nächstfolgende Speicherzelle, während der eben eingegebene Wert bei DATEN ablesbar ist.

d) Ist das Programm vollständig eingegeben, muß man den Eingabemodus wieder verlassen.
Dies geschieht durch einmaliges zusätzliches Drücken von

[EX]

Beispiel: Das folgende kleine (nicht funktionale) Programm soll eingegeben werden. Anfangsadresse soll 3C70 sein.

Beispiel: Das folgende kleine (nicht funktionale) Programm soll eingegeben werden. Anfangsadresse soll 3C70 sein.

00
06
37
23
76

a) [SET] [4/P] [3/I] [C] [7/X] [0/'] [EX]
b) [INPUT]

c) [0/'] [0/'] [EX]
[0/'] [6/Y] [EX]
[3/I] [7/X] [EX]

d) [2] [3/I] [EX]
[7/X] [6/Y] [EX]
[EX]

3.2.5.6. Ausgabe von Speicherbereichen

Wie bei der Eingabe ist auch bei der Ausgabe längerer Programme eine Bedienungs-Erleichterung vorgesehen. Auch hier ist lediglich die Eingabe der Anfangsadresse notwendig, das Weiterschalten geschieht dann automatisch.

a) Zuerst wird der Befehlszähler auf die Anfangsadresse des gewünschten Speicherbereichs gesetzt und der Wert dieser Zelle abgerufen

[SET] [4/P] [adresse] [EX]
[DIS] [MEM] [EX]

Der Inhalt dieser Zelle wird bei angezeigt, während die zugehörige Adresse bei ADRESSE steht.

b) Soll nun der Inhalt der nachfolgenden Zelle ausgegeben werden, wird einfach [IDM] gedruckt.

Dies hat zur Folge, daß der Befehlszähler um 1 erhöht wird und der Inhalt der damit bezeichneten Speicherzelle bei DATEN erscheint.
Durch fortgesetztes Drücken von IDM können somit z.B. auch längere Programme durchgesehen und überprüft werden, da nur eine Taste zu betätigen ist, während die Speicher-Adresse und ihr Inhalt jeweils gleichzeitig angezeigt werden.

Beispiel: Das vorher angegebene Programm befinde sich im Speicher mit Anfangsadresse 3C70 und soll schrittweise angezeigt werden.

a) [SET] [4/P] [3/I] [C] [7/X] [0/'] [EX]
[DIS] [MEM] [EX]

			ADRESSE	DATEN
Danach ergibt sich:			3C70	00
b)	[IDM]	→	3C71	06
	[IDM]	→	3C72	37
	[IDM]	→	3C73	23
	[IDM]	→	3C74	76

3.2.5.7. Starten eines Anwenderprogramms;

Dazu setzt man den Befehlszähler auf die Anfangsadresse des Programms und drückt daraufhin die Taste.

START

Solange das Anwenderprogramm bearbeitet wird erlöschen alle Anzeigeelemente.

Beispiel: Das Anwenderprogramm steht im Speicher mit der Anfangsadresse 3C66. Sie wollen es starten

SET 4/P 3/I C 6/Y 6/Y EX
START

Die Anzeigen erlöschen und leuchten erst nach Beendigung des Anwenderprogramms wieder auf.

3.2.5.8. Programmausführung im Einzelschrittverfahren ("single step")

Hierfür setzen Sie Ihren Befehlszähler auf die gewünschte Adresse im Programm und betätigen anschließend die Taste

STEP

Es wird genau ein Befehl ausgeführt. Bei ADRESSE erscheint der nun aktuelle Stand des Befehlszählers.

Beispiel: Gegeben sei wieder Beispiel 3.2.5.6
Der Befehlszähler sei bereits auf 3C70 gesetzt.

		ADRESSE
		3C70
STEP	→	3C71

Bitte machen Sie sich an dieser Stelle klar, daß **ein Befehl** im System Z80 **ein, zwei, drei oder vier Bytes** umfaßt. Der Befehlszähler (dieser Name ist also irreführend!) zählt jedoch Speicheradressen, also einzelne Bytes. Beim Einzelschrittverfahren springt der Befehlszähler bei jedem Betätigen der Taste STEP um 1, 2, 3 oder 4, je nachdem, ob die Ausführung 1, 2, 3 oder 4 Bytes langen Befehls durch die Betätigung ausgelöst wurde.

3.2.5.9. Setzen und Löschen eines Haltepunkts (Breakpoint)

(nur im RAM möglich)

a) Das Setzen eines Haltepunkts auf eine bestimmte Programmadresse wird durch folgende Tastenfolge erreicht:

BR adresse EX

"adresse" ist dabei durch den gewünschten Wert zu ersetzen.

Beispiel: In einem Anwenderprogramm, das im Speicher von Adresse 3C66 bis 3D66 untergebracht ist, soll zu Prüfzwecken ein Haltepunkt auf Adresse 3C92 gesetzt werden:

[BR] [3/I] [C] [9/L] [2] [EX]

Wird das Anwenderprogramm gestartet und erreicht der Befehlszähler diese Adresse, so wird das Programm angehalten. Sichtbar wird dies durch das Wiederaufleuchten der Anzeigeelemente.
Eine weitere Abarbeitung des Programms kann an dieser Stelle mit START oder STEP ausgelöst werden.

b) Das Löschen eines Haltepunktes kann auf zweierle Arten erfolgen

- während der Anwender-Programmabarbeitung durch START oder STEP, wenn das Programm den Haltepunkt erreicht hat

- durch die Tastenfolge

  [BR] [EX]

  wenn der Mikrocomputer nicht mit der Abarbeitung des Anwenderprogramms beschäftigt ist.

4. Entwicklung eines Mikrocomputer-Anwender-Systems

4.1. Entwicklungsziel

Soll ein Gerät - sei es ein Spielautomat, ein Meßgerät oder eine Kaffemaschine auf der Basis eines Mikrocomputers aufgebaut werden, muß sich der Geräteentwickler folglich mit der Mikrocomputersystem-Realisierung befassen.
Aus Bild 2, das die prinzipielle Struktur eines Mikro-Computersystems zeigt, wird klar, daß dabei außer einer Schaltungs- ("Hardware") entwicklung das Erstellen eines anwendungsspezifischen Programms ("Software") nötig wird, wobei übrigens die Hardwareentwicklung auf einen Bruchteil des in konventioneller Technik Erforderlichen reduziert ist. Um dem Entwickler wirtschaftliches Arbeiten zu ermöglichen, bieten potente Anbieter Standard-Lösungsteile an (das Fahrrad braucht ja auch nicht dauernd neu erfunden zu werden).

Auf dem Gebiet der **Hardware** sind dies:

- komplette Computerschaltungen, die in Form von Bausätzen ("Kits") zum Selbstzusammenlöten oder fertig bestückter und geprüfter Computerplatinen geliefert werden und
- Schaltungsbeispiele, die in Form von Applikationsbeispielen veröffentlicht und vom Anwender nachgebaut werden können.

Selbstverständlich kann mit solchen Hardware-Standards nur in wenigen Fällen die gesamte hardwareseitige Problematik der Geräteentwicklung abgedeckt werden:

So kann beispielsweise in einer Aufzugsteuerung auch der Einsatz eines Mikrocomputers den Aufbau der Elektromotor-Ansteuerung nicht ersparen.

Ein gewisser, wenn auch im Vergleich zu herkömmlichen Systemen geringer Hardware-Entwicklungsaufwand bleibt also auch im Mikrocomputersystemen nötig.

Zur Erleichterung der **Software-Entwicklung** ist eine Bibliothek von universell einsetzbaren Programmteilen (Unterprogramme, Makros) verfügbar, die es dem Anwender ersparen, häufig vorkommende Teilprobleme wie Gleitkommaoperationen, Sortierungen, Zeitschleifen usw. selbst zu lösen. Diese Programme können gegen eine Schutzgebühr bezogen werden. Der anwenderspezifische Rest des Programmes muß für jedes Problem eigens entwickelt werden.

Es soll an dieser Stelle angemerkt werden, daß bei Ausnutzung der erwähnten herstellerseitigen Unterstützung je nach Problem der

Softwareanteil = 50 ... 90% der Gesamt-System-Entwicklungskosten

beträgt.

Da dies bei mittleren und größeren Systemen erhebliche Aufwendungen darstellt, ist einsichtig, wie stark die Leistungsfähigkeit von Programmierhilfen auf den Gesamtentwicklungsaufwand Einfluß nehmen. Das Mikrocomputerentwicklungssystem, das hierbei höchste Rationalisierung erlaubt, wird daher im folgenden Abschnitt noch eingehend beschrieben.

An dieser Stelle soll zunächst noch das prinzipielle Vorgehen bei der Realisierung eines Mikrocomputersystems erläutert werden.

4.2. Wie geht man bei der Mikrocomputersystem-Entwicklung vor?

Die Entwicklung eines Mikrocomputersystems folgt im Prinzip dem Schema von Bild 57.

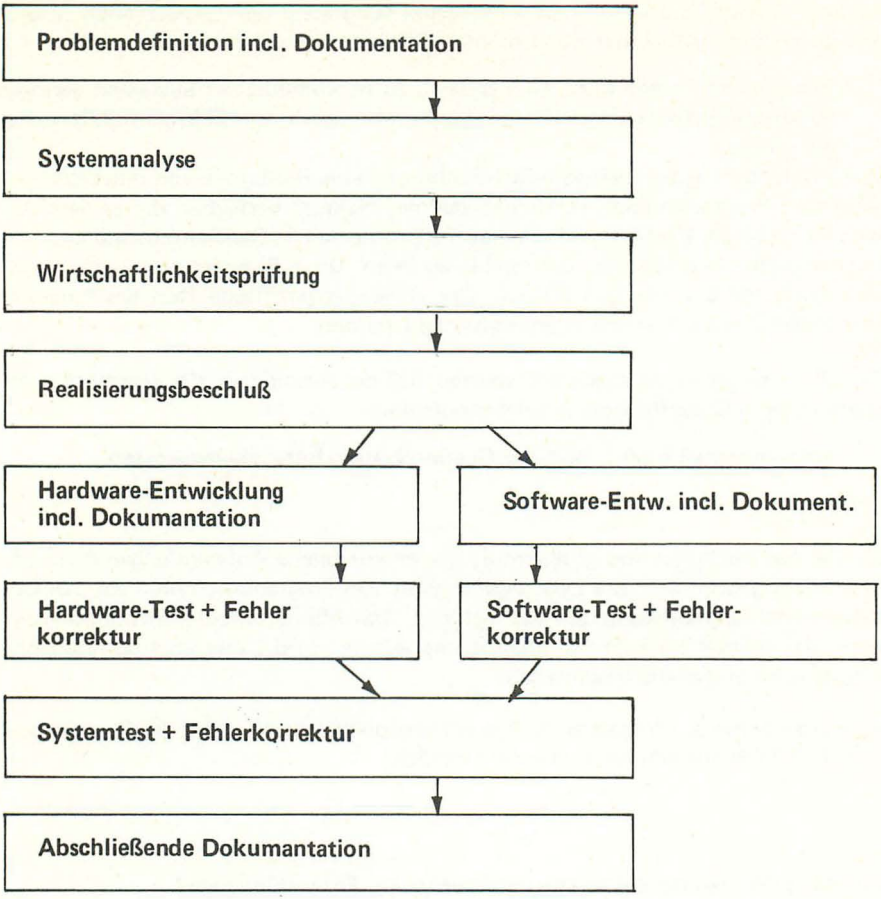


Bild 57: Enwicklungsablauf

In der **Problemdefinition** werden im wesentlichen die Anforderungen an das zu realisierende Gerät untersucht und dokumentiert.

Hier müssen insbesondere die Erfordernisse des künftigen Käufers bzw. Marktes (z.B. über die Marketing-Abteilung) und die von der Firmenleitung gesetzten Randbedingungen einfließen, um Fehlentscheidungen und "Totgeburten" zu vermeiden.

In der **Systemanalyse** sind alle vom Datenanfall bis zur Ergebnisauswertung durchzuführenden Arbeitsphasen einzeln und in ihren gegenseitigen Verflechtungen zu untersuchen und auf ihre Eignung für eine maschinelle Erledigung zu überprüfen.

Sie geht von den Ergebnissen der Problemdefinition aus und hat die Klarstellung der Frage zum Ziel, **wie** das definierte Problem technisch und wirtschaftlich zu realisieren ist. Besonders bei DV-Projekten aus dem kommerziellen oder Verwaltungs-Bereich, bei denen meist vielfältige Verflechtungen zwischen dem DV-Bereich und den vor- und nachgeschalteten manuellen Arbeitsgängen der Fachabteilung bestehen, kann diese Arbeitsphase komplex und aufwendig werden.

Sie sollte von Mitarbeitern durchgeführt werden, die neben der entsprechenden Fach- und allgemeinen Organisationskenntnissen über spezielle Kenntnisse der EDV-Organisation und der Programmierung verfügen.
Im Anschluß an die Systemanalyse muß nun eine **Wirtschaftlichkeitsprüfung** unter Berücksichtigung der möglichen Alternativen durchgeführt und ein Definition **Realisierungsbeschluß** gefaßt werden.

Auch hierbei ist wie bei den letztgenannten und folgenden Punkten sorgfältige Dokumentation kein Zeichen von übertriebener Bürokratie, sondern eher von verantwortungsbewußtem Vorgehen. Schon manche Firmen (und nicht nur kleine!) sind an sogenannten "Hobby Projekten" - Projekte, die ohne ernsthafte Wirtschaftlichkeitsprüfung und Managerentscheidung unter der Hand betrieben wurden - ebenso ehrgeiziger wie kurzsichtiger Entwickler gescheitert.

Diese ersten 4 Aktivitäten sind im wesentlichen unabhängig von der Technik, in der das Gerät realisiert werden soll, also ob einer mechanischen, festverdrahteten elektronischen oder Mikrocomputer-Lösung der Vorzug gegeben wird, es soll hier jedoch eine Checkliste vorgeführt werden, die die Mikrocomputer-spezifischen Gesichtspunkte einer Problemdefinition im Detail berücksichtigt.
Wichtiges Hilfsmittel hierfür ist der Datenflußplan (Bild 58).

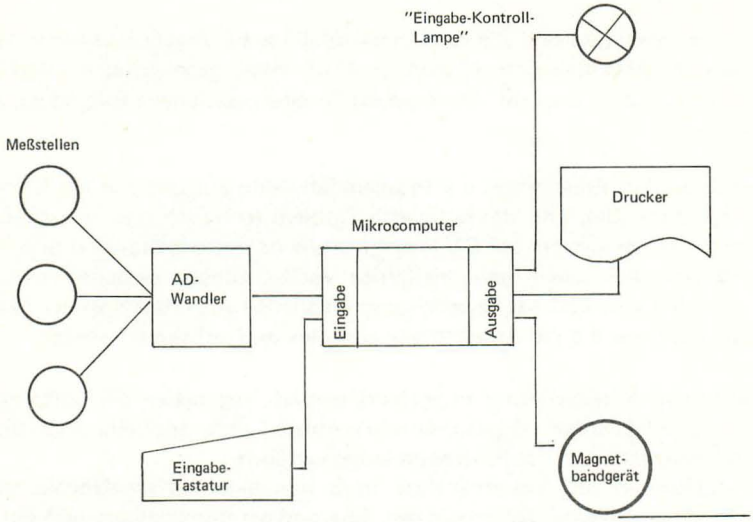


Bild 58: Datenfluß einer Meßwertüberwachungs- und Sammelstation

Der **Datenflußplan** ist die grafische Darstellung des Verarbeitungsablaufs eines Projekts. Er zeigt die zur Durchführung des Projekts erforderlichen Arbeitsgänge und deren Verbindungen auf. Da diese Verbindungen grundsätzlich durch die Weitergabe von **Daten** hergestellt werden, zeigt der Datenflußplan also den Weg oder den Fluß der Daten von Arbeitsgang zu Arbeitsgang.
Obwohl Datenflußpläne auch für manuelle Verarbeitungsverfahren aufgestellt werden können, werden sie in der Regel bei maschinellen Verfahren verwendet. Das schließt allerdings nicht aus, daß in Datenflußplänen auch manuelle Arbeitsgänge dargestellt werden.
Je nach der Abgrenzung der einzelnen Projekte steht am Anfang eines Datenflußplans die **Datenerfassung** oder die Übernahme der Daten aus einem anderen Projekt und am Ende die Weitergabe der Daten (z.B. Ergebnisse) an die für die Auswertung zuständige Fachabteilung, an die **Archivierung** für später durchzuführende Arbeiten oder an andere Projekte. Dazwischen sind die übrigen Arbeitsgänge in ihrer Bearbeitungsfolge aufgeführt.

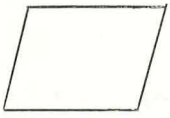
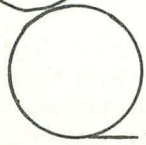
Die zur Übertragung der Daten verwendeten Datenträger sind ebenfalls Informationen, die im Datenflußplan dargestellt werden.

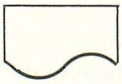
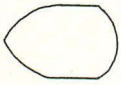
Zusammenfassend enthält der Datenflußplan Angaben über:

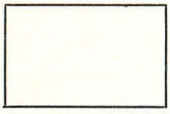
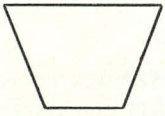
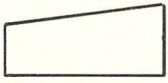
- o Quelle und Ziel der Daten bzw. Datenträger
- o Art, Anzahl und Reihenfolge der Arbeitsgänge
- o Art und Anzahl der Datenträger
- o Verbindungen und Abhängigkeiten der Arbeitsgänge
- o Art und Funktion der eingesetzten Geräte

Zur einheitlichen Gestaltung von Datenflußplänen werden genormte Symbole verwendet (DIN 66001). Sie werden in horizontaler oder vertikaler Richtung unter Hinzufügung von erläuternden Texten zu Datenflußplänen zusammenge

Folgende Symbole sind für die Mikrocomputertechnik relevant:

Symbole	Bezeichnung	Erläuterung
a) Symbole für Datenträger		
	Datenträger, allgemein	Wird insbesondere für Belege verwendet, wenn der Datenträger noch nicht endgültig feststeht, oder wenn verschiedenartige Datenträger eingesetzt werden.
	Lockarte	
	Lochstreifen	Zur näheren Kennzeichnung werden Streifenart und/oder Streifenbezeichnung angegeben
	Magnetband	Zur näheren Kennzeichnung werden Bandart und Dateibezeichnung angegeben

Symbol	Bezeichnung	Erläuterung
	Plattenspeicher (Festplatte oder Floppy Disk)	Zur näheren Kennzeichnung wird die Datenbezeichnung (Dateiname, Dateiart)
	Beleg oder Liste	Wird meist für Schriftstücke verwendet, die von einem Computersystem erstellt werden (z.B. Ausdrucke). Es kann aber auch für maschinenlesbare Belege verwendet werden.
	Anzeige	Wird für optische und akustische Anzeigegeräte verwendet (z.B. alphanumerische oder graphische Bildschirmanzeige).

Symbol	Erläuterung	Bezeichnung
b) Symbole für Bearbeitungsarten		
	Bearbeiten, allgemein	Wird für alle Bearbeitungsarten verwendet, für die es keine speziellen Sinnbilder gibt (z.B. Datenerfassung, Berechnung, Datenverarbeitung).
	Eingreifen von Hand	Bearbeitung ohne Verwendung maschineller Hilfsmittel (z.B.Durchführung von Korrekturen in Listen).
	Eingeben von Hand	Wird für die manuelle Eingabe von Daten verwendet B. Dateneingabe über Tastatur).

Auf die **Hardwareerstellung** brauchen wir hier nicht näher einzugehen; der periphere Teil (Leistungsschalter, Meßfühler, Signalübertragung) wird genauso ausgeführt wie in konventioneller Technik, für den Computer selbst stehen wie erwähnt weitgehend Standardlösungen zur Verfügung.

Der wichtigste Entwicklungsabschnitt, die **Software-Entwicklung,** kann parallel zur Hardwareerstellung folgen. Zweckmäßig ist eine gewisse personelle Spezialisierung (Hardware-Spezialist, Software-Spezialist); der Einsatz reiner "Programmierer" (wie in der konventionellen Datenverarbeitung üblich) ist jedoch nicht sinnvoll, da bei Mikrocomputersystemen Hard- und Software eng miteinander verflochten sind, daß ein Mikrocomputer-Hardware-Spezialist nicht ohne Programmierkenntnisse und ein Mikrocomputer-Softwarespezialist nicht ohne Schaltungskenntnisse effizient arbeiten kann. Den Entwicklungsablauf von Bild 57 wollen wir an dieser Stelle (Entwicklung des Anwenderprogramms) weiter detaillieren. (Bild 59):

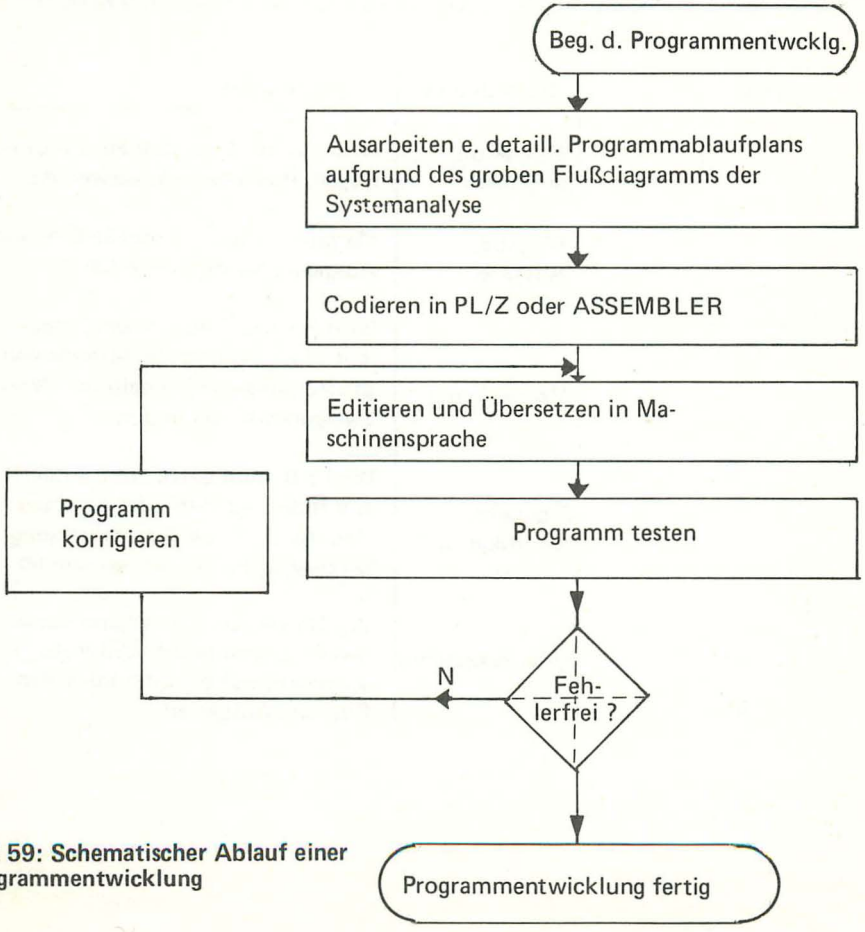
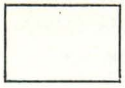
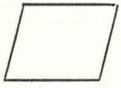
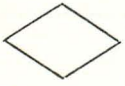
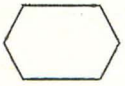
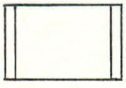


Bild 59: Schematischer Ablauf einer Programmentwicklung

Der Entwickler nimmt sich also zunächst den groben Funktionsablaufplan vor, der im Rahmen der Systemanalyse erstellt worden war und untergliedert die einzelnen "Kästchen" mit Rücksicht auf den zu verwendenden Mikroprozessor weiter; der resultierende dataillierte **Programmablaufplan** fächert das Gesamtproblem "Programmerstellung" in Stufen auf, so daß der Programmierer nicht gleichzeitig an die Programmlogik und die Realisierung dieser Logik durch konkrete Befehle aus dem verfügbaren Befehlsvorrat denken muß und stellt so sicher, daß die Übersicht nicht verloren geht.

Während der organisatorische Verarbeitungsablauf eines DV-Projekts durch den Datenflußplan veranschaulicht wird, werden im **Programmablaufplan** die einzelnen innerhalb eines Programms von der Datenverarbeitungsanlage der Reihe nach zu lösenden Schritte grafisch dargestellt. Ein Programmablaufplan stellt das logische Verarbeitungsschema, insbesondere hinsichtlich der im Programm enthaltenen Verzweigungen dar.

Analog zum Datenflußplan gibt es hier formale Konventionen (DIN 66001).

Symbol	Bezeichnung	Erläuterung
	Operation, allgemein	Wird z.B. für Operationen wie Übertragen, Rechnen usw. verwendet.
	Eingabe, Ausgabe	Maschinelle bzw. manuelle Ein- und Ausgabe über periphere Geräte.
	Verzweigung	Wird zur bedingten Programmablaufsteuerung und zur Abfrage von programmierten "Schaltern" (Verzweigungen) verwendet.
	Programmmodifikation	Wird z.B. zum Setzen und Ändern von Indexregistern oder sonstigen Registern, die zur Adreßrechnung herangezogen werden, verwendet.
	Unterprogramm	Zur Darstellung von abgeschlossenen Programmteilen (Unterprogrammen) mit ein oder mehreren Ein- und Ausgängen.

Symbol	Bezeichnung	Erläuterung
	Grenzstelle	Zur Darstellung von Anfang, Ende oder Zwischenhalt.
	Übergangsstelle	Zur Darstellung von Verbindungsstellen (Konnektoren) innerhalb eines Diagramms.
	Ablauflinie	Vorzugsrichtungen sind: a) von oben nach unten b) von links nach rechts
	Zusammenführung	Es ist zweckmäßig, am Zusammenführungspunkt eine Pfeilspitze zu zeichnen.

Eine zweckmäßige Methode zur übersichtlichen Gestaltung von Programmablaufplänen ist übrigens die Gliederung in **Haupt-** und **Unterprogramme (Unterpläne** und **Unterroutinen)**.
Ein in dieser Weise gegliederter Programmablaufplan enthält ein Hauptdiagramm mit den für die Steuerung des Programmablaufs wesentlichen Teilen sowie den Hinweisen auf durchzuführende Unterprogramme (Sinnbild ⊏⊐) und eine entsprechende Zahl von Unterdiagrammen (z.B. für die Ausgabe der Ergebnisse).

Diese Gliederung eines Programms ist nicht nur für die Erstellung des Programmablaufplans von besonderer Bedeutung, sondern erleichtert auch die anschließend durchzuführende Codierung. Beispielsweise können die einzelnen Unterprogramme von verschiedenen Programmierern weitgehend unabhängig kodiert werden.

Ist nun der Programmablaufplan schriftlich fixiert und auf seine Richtigkeit geprüft, kann das "Codieren" erfolgen, d.h. das Umsetzen der logischen Funktionsbausteine in **Anweisungen** an den Computer bzw. **Befehle**.
Vor Beginn dieser letzten Tätigkeit muß die Entscheidung gefallen sein, in welchem "Code" das Programm geschrieben werden soll.
Bekanntlich muß dem Computer das Programm in Maschinencode zugeführt werden d.h. der CPU muß eine Folge von binären Nullen und Einsen ("bits") angeboten werden, wobei jedem Befehl, den die CPU akzeptiert, eine bestimmte bit-Kombination entspricht. Da 1 Befehl aus 8 ... 40 bits besteht (System Z80), ist ein Niederschreiben des Programms wegen Unanschaulichkeit und Unübersichtlichkeit dieser Darstellung unsinnig und Fehlerkorrekturen viel zu langwierig. Man schreibt die Programme daher

* in einer höheren Programmiersprache (z.B. PL/Z, Basic, Cobol) nieder, wobei ein oder mehrere Maschinen-Befehle durch ein Symbol dargestellt werden (z. B. das Pluszeichen "+" für eine Addition) oder

* in Assembler-Sprache, in der **ein** Maschinenbefehl durch **ein** anschauliches Textkürzel (z.B. "ADD" für "addiere") ersetzt ist und die einzelnen Speicherplätze durch frei wählbare symbolische Namen bezeichnet ("adressiert") werden können (z.B. "EMMA", "ZIEL" usw.).

Unter Berücksichtigung der Regeln ("Konventionen") der höheren Programmiersprache (PL/Z bzw. ASSEMBLER-Sprache), die in den entsprechenden Sprachbeschreibungen übersichtlich fixiert sind, schreibt der Entwickler nun die dem Programmablaufplan äquivalente Folge von Befehlen auf.
Der nächste Schritt in der Programmerstellung muß jetzt die Umwandlung dieses Programms in die für den Prozessor "verständliche" Form sein ("Übersetzung"); hierbei bezeichnet man ein Programm, das in einer höheren Programmiersprache oder in ASSEMBLER-Sprache vorliegt, als Primärprogramm oder Quellenprogramm und das Ergebnis der Übersetzung als Objektprogramm.
Da die Übersetzung selbst eine rein mechanische und für den Menschen zeitaufwendige Arbeit ist, setzt man hierfür bei Programmen mit mehr als 100 Befehlen vernünftigerweise einen Computer (übrigens natürlich nicht den Anwender Mikrocomputer!) ein, der im übrigen noch den Vorteil hat, daß er die Übersetzungsarbeit fehlerlos ausführt und sogar in vielen Fällen auf Fehler im Primärprogramm aufmerksam macht ("Fehlermeldung").
Die Fähigkeit, eine solche Übersetzung durchführen zu können, erhält der Computer durch ein entsprechendes Übersetzungsprogramm, das im Falle der Verwendung einer Höheren Programmiersprache (Bild 60) **Compiler** heißt (z.B. PL/Z Compiler) und im Falle der Übersetzung aus ASSEMBLER-Sprache als **Assembler-Programm** (Bild 61) bezeichnet wird. Man muß also bei der Benützung des Wortes "Assembler" aufpassen, ob die Sprache oder das Programm gemeint ist.
Bei beiden Arten der Übersetzung ist also das **Primärprogramm** die Eingabe für den Computer, der die Übersetzung durchführt, und das **Objektprogramm** das Ergebnis, das dieser Computer ausgibt (Bilder 60 und 61). Der Unterschied ist der, daß **Assembler-Programme** aus einer zu übersetzenden Quellenanweisung genau **einen** Maschinenbefehl erzeugen, während **Compiler** die Funktion der Quellenanweisung durch eine **oder mehrere** Maschinenanweisungen formulieren.

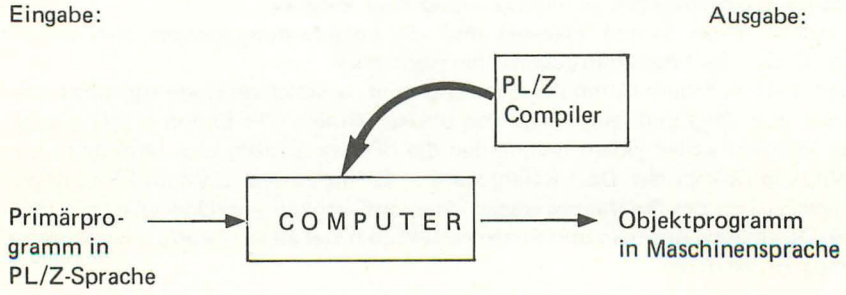


Bild 60: Compilierung (Übersetzung eines PL/Z-Programms)

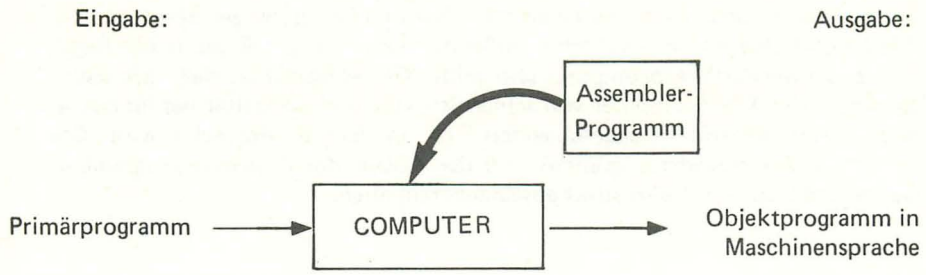


Bild 61: Assemblierungsvorgang (Übersetzung eines ASSEMBLER-Programms)

Wir müssen nun noch die Frage klären, wie man das Primärprogramm in den Computer bringt, das ja zunächst nichts anderes ist als Schrift auf einem Blatt Papier: Das Primärprogramm muß **eingegeben** werden.
Da ein Computer ohne Programm "dumm" ist (also auch nicht in Lage ist, eine Eingabe zu "verstehen", d.h. in einer bestimmten Form zu akzeptieren), ist schon für die Eingabe des Primärprogramms ein eigenes Dienstprogramm nötig, der sogenannte **Editor** (Bild 62).

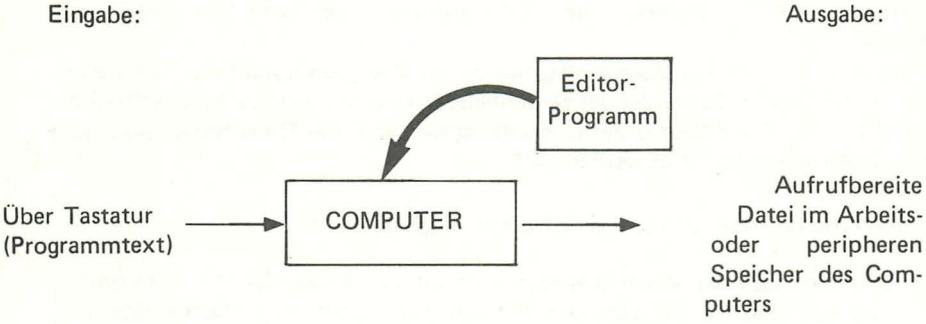


Bild 62: Editiervorgang

Mit Hilfe des Editors, **der selbst nicht von der verwendeten Programmiersprache abhängt**, kann man Programme und sonstige beliebige Texte dem Computer eingeben und zwar nach Wunsch in dessen Arbeits- oder peripheren Speicher (z.B. Floppy Disk). Diesen Vorgang bezeichnet man allgemein als "Dateiaufbau". Weiterhin können mit dem Editor Programmstellen aufgesucht und Korrekturen durchgeführt werden. Die fertige Datei, die das Primärprogramm enthält, kann nun abgespeichert werden; das Übersetzungsprogramm kann der Entwickler dann jederzeit anweisen, diese Datei als Eingabe (Primärprogramm) zu verwenden und in den Objekt-Code umwandeln.

Wir haben bisher noch keine Feststellungen über den Computer getroffen, der das Editieren und Übersetzen durchführt, außer der Bemerkung, daß dies in der Regel nicht der Anwender-Mikrocomputer sein sollte. Grund hierfür ist, daß - wie schon ausgeführt - der Mikrocomputer von seiner Struktur und Kapazität her für das jeweilige Anwendergerät "maßgeschneidert" ist, in dem er eingesetzt wird. Die Erstellung des Anwenderprogramms und der Ablauf des Anwenderprogrammes ("Gerätefunktion") sind also strikt auseinanderzuhalten.

Selbstverständlich kann auch für Programmentwicklungszwecke ein Mikrocomputer "maßgeschneidert" werden, der dann für die Aufgabe der Anwenderprogrammerstellung besonders gut geeignet ist.
Dies ist beispielsweise realisiert in Form des Z80-Mikrocomputer-Entwicklungssystems, das wir im folgenden noch näher kennenlernen werden.

Die Dienstprogramme, die auf diesem Spezial-Mikrocomputersystem ablaufen, nennt man "Residente Software".
Die Alternative zum Spezialcomputer ist die Verwendung eines Universalcomputers ("General Purpose"), wie Anlagen der Serie IBM 370 als "Gastrechner" (=" Host-Computer"). Auch mit ihnen sind Editier- und Übersetzungsarbeiten möglich. Hierzu sind allerdings wieder spezielle Programme nötig, die man **"Cross-Software"** nennt, da sie sozusagen das softwaremäßige "Überwechseln" von einem Computer (dem Gastrechner) auf einen anderen (eben dem Mikrocomputer) erlauben.
In jedem Fall — ob bei Übersetzung mit einem Mikrocomputer-Entwicklungssystem, einem Großrechner oder der zeitraubenden Übersetzung "zu Fuß" (="Handcoding") — ist das Ergebnis des Übersetzungsvorgangs das Objektprogramm, das nun als nächstes ausgetestet werden muß.

Für den Programmtest gibt es 3 verschiedene Möglichkeiten:

- ☐ Ablaufenlassen des Maschinenprogramms auf dem Anwender Mikrocomputer. Voraussetzung ist natürlich, daß die Anwenderschaltung zu Softwaretestbeginn fertiggestellt wurde. Nachteildes Verfahrens ist, daß der ganze Test ohne besondere Testhilfen durchzuführen und daher zeitraubend ist.

- ☐ Benutzung eines Universal-Computers, der mit einem hierfür entwickelten Testprogramm die Funktion des Anwender-Mikrocomputers und den Ablauf des Anwenderprogramms auf diesem Mikrocomputer nachbildet. Ein solches Programm heißt **Cross-Simulator** und wird nur dann verwendet, wenn die Programmentwicklung auf einem Universalcomputer durchgeführt wird.

Nachteile dieses Verfahrens:

- ☐ On-Line-Zugriff auf einen Universalrechner nötig
- ☐ der Programmtest erfolgt nicht in "Echtzeit" (="Real-Time"), da Gastrechner und Anwender Mikrocomputer zwei unterschiedliche Maschinen sind und

zwangsläufigerweise unterschiedliches Zeitverhalten haben. Daraus folgt, daß ein fehlerfreier Ablauf der Simulation nicht den fehlerfreien Ablauf des Anwenderprogramms auf dem Anwender-Mikrocomputer garantiert.

- Benutzung eines Mikrocomputer-Entwicklungssystems, auf dem das Anwenderprogramm auf Echtzeit (da der im Mikrocomputer-Entwicklungssystem und im Anwendersystem verwendete Mikroprozessor gleichen Typs ist) ablaufen und auf Richtigkeit geprüft werden kann.

Stellen sich beim Test - gleichgültig auf welche Weise er durchgeführt wurde - Fehler heraus, muß der Entwickler sie mit Hilfe des Editors im Primärprogramm korrigieren, das verbesserte Primärprogramm neu übersetzen und erneut testen, bis beim Programmtest keine Fehler mehr festzustellen sind. Als nächstes kann nun getestet werden, ob das getrennt getestete Programm (Software) und die getrennt getestete Schaltung (Hardware) "zueinanderpassen". Dieser Test ist - wegen der vielfältigen Fehlermöglichkeiten - relativ kompliziert und zeitaufwendig.

Aufgrund dieser Erkenntnis wurde ein spezieller Zusatz zum Mikrocomputer-Entwicklungssystem, ein sogenannter Emulations- und Test-Adapter (User's Hardware Interface) entwickelt, der Tests unter genau den Bedingungen erlaubt, unter denen das gesamte Anwendersystem arbeitet (Echtzeit); dabei verbindet der Adapter direkt den Musteraufbau des Anwenders Mikrocomputer mit dem Mikrocomputer-Entwicklungssystem.
Aus diesen Ausführungen ergibt sich nebenbei die Liste der Vorteile, die ein Arbeiten mit einem Mikrocomputer-Entwicklungssystem gegenüber anderen Verfahren (wie bei Universalrechner) mit sich bringt:

- Der Emulations- und Test-Adapter testet in Echtzeit, d.h. genau unter den gleichen Bedingungen wie das Anwendersystem
- Der Emulations- und Test-Adapter arbeitet direkt mit dem Versuchsaufbau des Anwendersystems zusammen
- Der Emulations- und Test-Adapter verlegt die gesamten Testmöglichkeiten des Mikrocompuer-Entwicklungssystems direkt in den Anwenderversuchsaufbau.

An dieser Stelle ist anzumerken, daß Tests einen wesentlichen Anteil der gesamten Entwicklungszeit (typisch 30 ... 80%) ausmachen. Gerade an dieser Stelle ist eine Rationalisierung der Entwicklungsarbeit durch Einsatz technischer Hilfsmittel besonders wirtschaftlich.
Stellen sich beim Systemtest Fehler heraus, sind diese in der Hard- und Software zu ändern. Bei der Korrektur von Systemfehlern ist außer evtl. Hardwareverbesserungen wie beim Softwaretest eine Korrektur des Anwenderprogramms mit Hilfe des Editors, Neuübersetzung und Wiederholung des Systems nötig.

4.3. Praktische Beispiele von Programmentwicklungen

4.3.1. Ein sehr einfaches Spielprogramm

Wir wollen nun zunächst einmal ein ganz kurzes Programm mit einer sehr einfachen Funktion beschreiben.

Mit ihm soll einfach eine Serie von 8 Lämpchen bzw. Leuchtdioden in einer bestimmten Geschwindigkeit zum Aufleuchten gebracht werden.
Sobald alle Lämpchen der Zeile aufgeleuchtet sind, sollen sie alle wieder verlöschen und das ganze Spiel soll endlos weiter ablaufen.
Der Datenflußplan ist in diesem Fall trivial (Bild 63) er beinhaltet außer dem Computer (z.B. eine Standardeuropaplatine Z80-ECB/C, die auch über Parallelschnittstellen verfügt) lediglich die Anzeige und eine Rücksetztaste zum Initialisieren des Computers; für den Fall, daß der Leser dieses einfache Spielchen nachbauen will, ist im Bild 64 Schaltplan und Layout gezeigt.

Den Programmablauf finden Sie in Bild 65

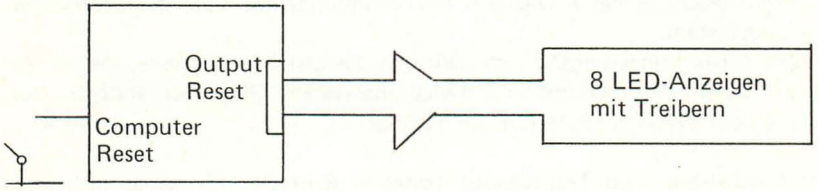


Bild 63: Datenflußplan fürs Lämpchen-Spielprogramm

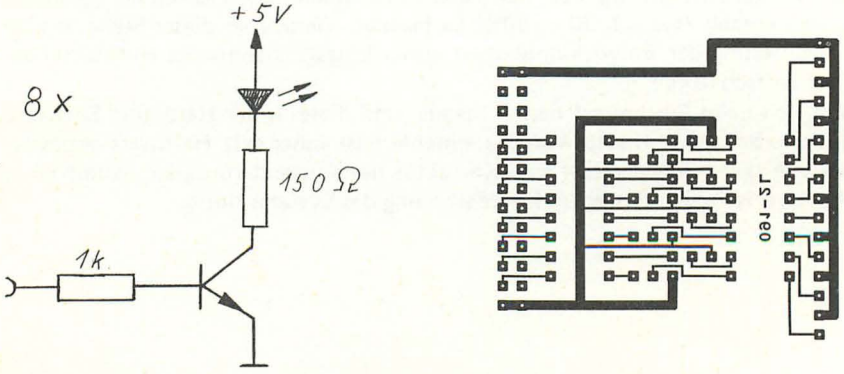


Bild 64: Schaltplan und Layout fürs Lämpchen-Spielprogramm

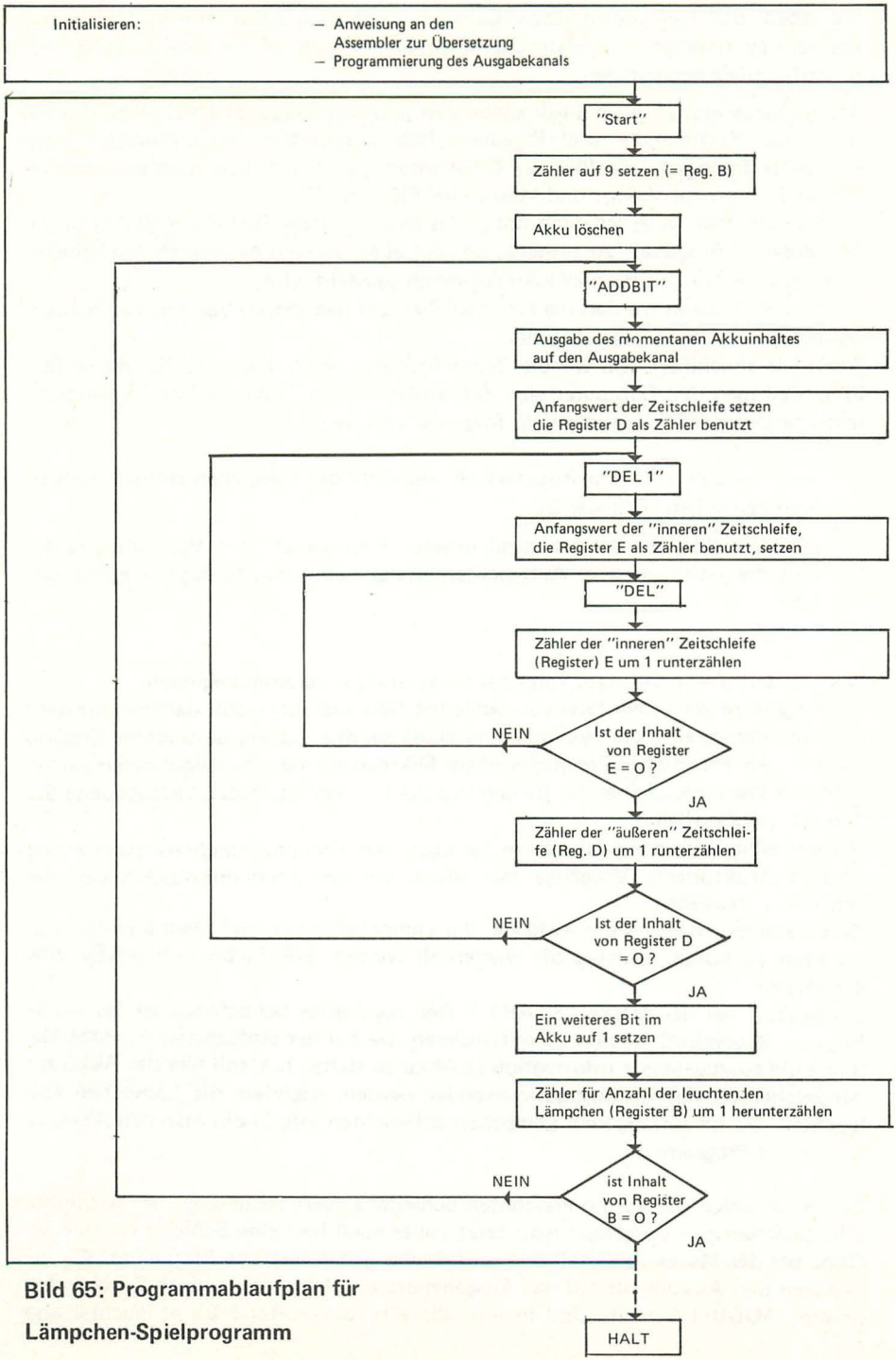


Bild 65: Programmablaufplan für Lämpchen-Spielprogramm

Sie sehen, daß hier wie in jedem Computerprogramm zunächst eine Anzahl Initialisierungsvorgänge einzuplanen sind, um am Anfang der Verarbeitung definierte Verhältnisse herzustellen.

Als nächstes gibt man im allgemeinen den programmierbaren Bausteinen des Systems die Kommando- und Parameter-Information, die deren Funktion oder Parameter festlegen, wie (bei der CPU) Interrupt - Mode, Stackpointer oder Betriebsart, Interrupt-Vektor und Masken bei PIO oder CTC.
In unserem Fall ist es lediglich nötig, das eine benutzte Port der Z80-PIO in die Betriebsart "Ausgabe" zu bringen, da von allen übrigen genannten Funktionen in unserem einfachen Beispiel kein Gebrauch gemacht wird.
Im Anschluß daran werden die für unser Problem benötigten Variablen auf die gewünschten Anfangswerte gebracht.
Wenn wir annehmen, daß wir die Zeitverhältnisse per Software realisieren wollen (d.h., daß man den Computer eine Zeit einfach durch Zählen seiner Taktimpulse selbst festlegen läßt), brauchen wir folgende Variablen:

* eine Variable (z.B. ein Register), die die Zahl der Lämpchen enthält, die aufleuchten sollen (Register B)
* eine Variable, die entsprechend unserer eben getroffenen Vereinbarung die Zeit die zwischen dem Aufleuchten zweier Lämpchen festlegt (Registerpaar DE).

Nach diesen Vorbereitungen kann das funktionale Programm beginnen.
Im Programm wird mehrfach von **Schleifen** Gebrauch gemacht: darunter versteht man Programmteile, die zyklisch wiederholt werden, bis ein bestimmtes Ereignis auftritt. Als ein solches Ereignis wird bei Mikrocomputern im Allgemeinen ein bestimmter Wert irgendeines der Bedingungsbits (="Flag") benutzt, vorzugsweise das Nullbit (="Zeroflag").
Mit der Schleifentechnik ist man in der Lage, dem Computer mehrere gleiche oder ähnlich strukturierte Vorgänge mit einem einzigen Programmstück (eben der Schleife) vorzugeben.
Da in unserem Beispiel die Aufgabe, die Lämpchen eines nach dem anderen aufleuchten zu lassen, beliebig oft wiederholt werden soll, bietet sich hierfür eine Schleife an.
Sie beginnt bei der Marke "START". Der zugehörige Schleifenzähler (es wurde Register B gewählt) ist nun zu initialisieren. Da bei der einfachsten Ausgabe-Methode die auszugebende Information in Akku zu stehen hat, soll hier der Akku zur Abspeicherung des Bitmusters verwendet werden, nachdem die Lämpchen aufleuchten. Da im Anfang kein Lämpchen aufleuchten soll, löscht man den Akku zu Beginn des Programms.

Das Hochzählen des vorhin erwähnten Schleifenzählers besteht aus der Wiederholung gleichartiger Vorgänge; man setzt daher auch hier eine Schleife ein. Sie beginnt bei der Marke ADDBIT und umfaßt das ganze restliche Programm. Wir geben nun den Akkuinhalt auf das Ausgabeport aus; beim erstmaligen Durchlaufen unserer ADDBIT-Schleife sind in ihm alle Bits rückgesetzt (=0), es leuchtet also

kein Lämpchen. Nach dem achten Durchlauf sind alle Bits "1". Sofort nach der Ausführung des Ausgabebefehls ist das gewünschte Bitmuster im Ausgaberegister des Ausgabebausteins gespeichert und sorgt dafür, daß die entsprechenden Lämpchen aufleuchten ohne daß der Prozessor weitere Prozessoraktivitäten (abgesehen natürlich von Ausgaben auf eben dieses Ausgabeport!) unsere Peripherie (also die Lämpchen) in irgendeiner Form beeinflußt.
Es bleibt uns jetzt eigentlich nur noch übrig, den Prozessor die Zeit ermitteln zu lassen, bis das nächste (bzw. nach Programmstart das erste) Lämpchen aufleuchten soll.
Wir wollen auch diesen Vorgang mit einer Schleife durchführen. Da ein Register nur 8bit lang ist (den Werten 0 ... 255), die für den Menschen wahrnehmbare Zeit jedoch in Sekunden liegt, genügt ein einziges solches Zählregister nicht (beim Zählvorgang werden ja Befehlsausführungszeiten gezählt, die nur einige wenige Mikrosekunden dauern).
Man braucht aus diesem Grund ein zweites Zählregister, das man in eine weitere Schleife (DEL 2) hochzählt, die in der ersten Schleife eingebettet ist.
Der Vorgang ist dann wie folgt:

Zunächst wird die erste Zählschleife auf ihren Anfangswert gesetzt und durchgeführt. Daraufhin wird der zweite Schleifenzähler auf seinen Anfangswert gesetzt und der erste Zählschritt in DEL 2 durchgeführt. Der DEL 2-Zählschritt wird so oft wiederholt, bis der Schleifenzähler von DEL 2 Null ist. Jetzt wird der erste Zählschritt von DEL 1 durchgeführt; es ist nun eine Zeit verstrichen, die der Ausführungszeit der Zählbefehle mal dem Anfangswert des DEL 2-Zählers entspricht.

Durch den Rücksprung nach DEL 1 wird nun die innere Schleife DEL 2 wieder neu begonnen und wie vorher ausgeführt. Der gesamte Vorgang wird sooft wiederholt, bis auch der Zähler DE auf Null heruntergezählt ist; die dann verstrichene Zeit entspricht dem Produkt aus der Ausführungsdauer der Schleifen DEL 1 und DEL 2.
Will man die genaue so erzeugte Verzögerungszeit berechnen, muß man selbstverständlich eventuelle Vorgänge berücksichtigen, die einmalig sind oder nicht bei jedem Zählvorgang vorkommen (z.B. das einmalige Initialisieren jeder Schleife oder Unterschiede in der Verarbeitungszeit bei positivem und negativem Ergebnis des Schleifenende-Abfrage-Befehls).

An dieser Stelle kann jetzt der Akkuinhalt verändert werden (also ein weiteres Bit gesetzt werden), um das Aufleuchten des nächsten Lämpchens vorzubereiten.

Außerdem wird zur einfachen Kontrolle der Anzahl der leuchtenden Lämpchen ein Zählregister (B) heruntergezählt, das gleichzeitig als Schleifenzähler für die Schleife ADDBIT fungiert, die soviele Male durchlaufen wird, wie Lämpchen zum Aufleuchten zu bringen sind.

Soweit alle Lämpchen zum Aufleuchten gebracht wurden, hat das Zählregister B den Wert Null und die Fortsetzung des Programms erfolgt bei der Marke "START".

Diese Verzweigung (="Sprung") zum Programmbeginn bewirkt, daß das gesamte Programm "ewig" ausgeführt wird, soweit nicht ein hardwaremäßiger Eingriff von außen erfolgt.
Wünscht man diese Verhaltensweise nicht, sondern möchte man stattdessen, daß die Programmausführung nach einem einmaligen vollen Durchlauf beendet wird (also sobald alle Lämpchen zum Aufleuchten gebracht wurden), müßte man wie Bild 65 gestrichelt gezeigt, einen HALT-Befehl einfügen.
Dieser HALT-Zustand des Prozessors kann über einen Interrupt wieder verlassen werden.
Zu diesem Zeitpunkt ist der wichtigste (und neben den Test wohl aufwendigste) Teil unserer Programmentwicklung abgeschlossen.

Es folgt nun das Codieren, d.h. das Formulieren der entwickelten Programmlogik in der entsprechenden Computersprache (in unserem Fall Z80-ASSEMBLER).
Das Ergebnis dieser Arbeit ist ein meist handgeschriebenes Programm, das nach dem Editieren und Übersetzen die im Bild 66 dargestellte Form eines **Listings** bekommt.

Dieses Listing wurde mit einem Z80-Entwicklungssystem erzeugt. Steht dem Entwickler ein solches Hilfsmittel nicht zur Verfügung, was meist bei Entwicklungen mit dem Z80-KIT der Fall ist, muß er eine äquivalente Übersetzung von Hand durchführen, d.h. in der Befehlsliste die zu den ASSEMBLER-Befehlen zugehörigen hexadezimalen Maschinencodes heraussuchen und den symbolischen Adressen (wie "START") feste hexadezimale Speicheradressen zuordnen. Das ASSEMBLER-Programm des Entwicklungssystems tut beides automatisch; in dem Listing Bild 66 finden Sie die Speicheradressen ganz links (also in der 1. Spalte) und die zugehörigen Inhalte (das sind im Programmspeicher Befehle und Konstantenwerte, die durch die Pseudobefehle DEFB, DEFW usw. festgelegt werden) in der nächstfolgenden Spalte.
In der Spalte "STMT" (="Statement") finden Sie die laufende Nummer der Quellenprogrammzeile; darauf folgen die symbolischen Assemblerbefehle, Marken und Kommentare.
Beachten Sie, daß den Kommentaren nie ein Maschinen-Codewort entspricht, d.h. sie kommen im Maschinenprogramm nicht vor und haben auf dieses auch keinen Einfluß.
Zur Codierung sind nur noch wenige Bemerkungen zu machen, da unser Programmablaufplan bereits so detailliert war, daß praktisch jeder Aktivität ein und nur ein ASSEMBLER-Befehl entspricht und man das Quellenprogramm daher praktisch aus dem Z80-CPU-Befehlssatz "ablesen" kann.

Zur Initialisierung muß man dem Assembler sagen, auf welche physikalische Speicherstelle er den Anfang des ablauffähigen Maschinenprogramms legen soll (Zeile 4, ORG 3C50H).
Die dort angegebene Adresse hängt von der Hardware ab, in der das Programm ablaufen soll; die hier verwendete Adresse 3C50 wurde gewählt, um das Programm in den RAM-Bereich des Z80-KIT einschreiben zu können.

Anwenderprogramme, die in Festwertspeicher gebracht werden sollen, wird man meist ab Adresse 0 beginnen lassen (ORG 0H), weil ihr Anfang dann beim Anlegen der Speisespannung automatisch aufgefunden wird, da der Befehlszähler der CPU in diesem Fall auf 0 gesetzt wird.

```
                                  LAMP   LISTING                          PAGE   1
 LOC    OBJ CODE    STMT SOURCE STATEMENT                                  ASM 3.0

3C50                  1  ORG 3C50H
                      2 ;VORFUEHR
                      3 ;PROGRAMM ZUR ANSTEUERUNG VON LEDS
                      4 ;
                      5 ;
                      6 PBC   EQU 0BH      ;I/O PORT B CONTROL  INFO-ADR
                      7 PBD   EQU 09H      ;I/O PORT B DATEN - ADR
                      8 ;
3C50    3E0F          9  LD     A,0FH      ;PORT B  IN
3C52    D30B         10  OUT    (PBC),A    ;OUTPUT MODE
                     11
                     12 START:
3C54    0609         13  LD     B,9        ;LAMPENZAEHLER INITIALISIEREN
3C56    3E00         14  LD     A,0        ;AUSZUGEB. BYTE = 00
                     15
                     16 ADDBIT:
3C58    D309         17  OUT    (PBD),A    ;NAECHSTES BYTE AUSGEBEN
                     18 ;
3C5A    16FF         19  LD     D,255      ;AEUSS. ZAEHLSCHLEIFE INITIAL.
                     20 DEL1:
3C5C    1E32         21  LD     E,50       ;INNERE ZAEHLSCHLIFE INITIAL
                     22 ;
                     23 DEL2:
3C5E    1D           24  DEC    E          ;INNERE SCHLEIFE RUNTERZ.
3C5F    20FD         25  JR     NZ,DEL2
                     26 ;
3C61    15           27  DEC    D          ;AEUSSERE SCHL. RUNTERZ.
3C62    20F8         28  JR     NZ,DEL1
                     29 ;
3C64    37           30  SCF               ;CARRY FLAG SETZEN
3C65    CB17         31  RL     A          ;ACCU NACH LINKS SCHIEBEN UND
                     32 ;                   CARRY BIT AUF BIT 0 VON ACCU
                     33 ;
3C67    10EF         34  DJNZ   ADDBIT     ;LAMPENZAEHLER RUNTERZAEHLEN
                     35 ;
3C69    C3543C       36  JP     START
                     37 ;
                     38 END

   0 ASSEMBLY ERRORS
```

Bild 66: Listing des Lämpchen-Spielprogramms

Die "Pseudobefehle" (=Anweisungen an das Übersetzerprogramm) sagen aus, daß die im Quellen-Programm verwendeten symbolischen Portoadressen PBC und PBD für das ablauffähige Maschinenprogramm jeweils durch die Hexadezimalwerte 0B und 09 zu ersetzen sind.

Das Initialisieren des Z80-PIO-Bausteins geschieht über die Befehle 9 und 10. Hier wird zunächst der Hexadezimalwert 0F des Kommandowortes, das den Z80-PIO-Baustein in die Betriebsart "Ausgabe" bringt (s. Z80-PIO-Spezifikation oder Manual) und über den Ausgabebefehl OUT an den Steuerteil des PIO's ausgegeben, der mit PBC (=0BH) adressiert ist.

Wir können nun unseren Schleifenzähler B mit dem Anfangwert laden (der um 1 höher ist als die Anzahl der durchzuführenden Durchläufe, da er gleich zu Pro-

grammbeginn um 1 heruntergezählt wird) den Akku auf den Anfangswert Null bringen (Zeile 14) und an den Datenteil des PIO-Ports B der Akkuinhalt übertragen (Zeile 17).
Bei der Berechnung der erforderlichen Gesamtverzögerungsdauer der Zeitschleifen DEL 1 und DEL 2 kommt man auf ca. 13000 Zählschritte; man setzt daher den Anfangswert von Schleifenzähler DEL 1 (Register D, Zeile 19) auf seinen Maximalwert 255 und den des Schleifenzählers DEL 2 (Register E, Zeile 21) auf 50 (≙12750 Zählschritte).
In Zeilennummer 24 wird das Zählregister E um 1 heruntergezählt (="dekrementiert"). Die Abfrage, ob das Schleifenende erreicht ist, geschieht für DEL 2 mit dem Befehl der Zeile 25.
Dieser Befehl bewirkt einen Rücksprung auf die Marke DEL 2 für den Fall, daß der Registerinhalt noch nicht gleich null ist (JR NZ, DEL 2) bedeutet:

("Jump relative on Not Zero to DEL 2 = Relativer Sprung bei Rechenergebnis = Null auf die Adresse DEL 2").
An dieser Stelle wird einer der vielen Vorteile der Z80-CPU offensichtlich. Es sind hier relative Sprünge möglich, das sind Befehle, bei denen nicht die volle 2Byte= 16bit - Sprungzieladresse angegeben werden muß, sondern nur eine 1Byte-Distanz (= 127 Adressen), die sich auf den momentanen Stand des Befehlszählers bezieht. Dadurch wird Speicherplatz gespart.

Bei absoluten Sprüngen, die eine Verzweigung zu jeder beliebigen der 64k - Speicherstellen ermöglichen, ist eine Angabe der vollen 2Byte-Speicheradresse im Maschinencode erforderlich. Da die meisten in der Praxis vorkommenden Sprungbefehle nur "kurze" Distanzen im Programmspeicher zu überbrücken haben, kann dieser überflüssige Programmspeicher- und Verarbeitungszeit-Aufwand durch Verwendung relativer Sprünge eingespart werden.

In analoger Weise zu DEL2 wird nun die Schleife DEL1 vom Befehl in Zeile 27 heruntergezählt und in Zeile 28 darauf getestet, ob sie bereits fertig abgearbeitet ist.

Die nächste Aufgabe des Programms war, den Akkuinhalt so zu verändern, daß das nächstfolgende bit gesetzt wird, um die nächste Lampe zum Aufleuchten zu bringen. Dies erfolgt in den Befehlen der Zeilen 30 und 31.
Benutzt wird hierzu der links-Rotationsbefehl, bei dem der Registerinhalt bit für bit um eine Stelle nach links verschoben wird, wobei der Inhalt des Übertragsbits (="Carryflag") mit berücksichtigt (also von rechts her in das Register hineingeschoben) wird.
Da unser Ziel ist, immer ein weiteres bit auf 1 zu setzen, um ein weiteres Lämpchen aufleuchten zu lassen, müssen wir dafür sorgen, daß das jeweils in das Register hineingeschobene bit des Carry-Flags zu diesem Zeitpunkt immer gleich Eins ist. Dies geschieht durch den Befehl SCF (="Set-Carry-Flag") in Zeile 30.
Bitte erinnern Sie sich, daß die eigentliche Ausgabe des Registers-(in diesem Fall Akku-) Inhaltes zu einer anderen Programmstelle erfolgt, nämlich in Zeile 17. Bis zur Ausführung dieses Befehls bleibt der Wert der Ausgabe (=aufleuchtende Lämpchen) unverändert.

Dies hat selbstverständlich eine Zeitverschiebung zur Folge, die allerdings in unserem zeitunkritischen Fall, bei dem es auf ein paar Mikrosekunden nicht ankommt, völlig unerheblich ist.
Für die nun anstehende Schleife ADDBIT wollen wir wieder einen besonderen Befehl anwenden, der die Anwenderfreundlichkeit des Z80-Systems in hohem Maße verdeutlicht.
Dies ist der Befehl DJNZ (="Decrement and Jump on Not Zero"), der eine vollständige 1Byte-Schleifenbehandlung erlaubt (Zeile 34); er subtrahiert 1 vom Momentanwert des Zählregisters B und springt auf die im Operanderfeld angegebene Adresse (hier ADDBIT), falls B den Wert Null noch nicht erreicht hat. Andernfalls (also wenn der Zähler B auf Null heruntergezählt ist) verläßt er die Schleife, indem er einfach den nächstfolgenden Befehl ausführen läßt.
Damit ist unser Programm schon ziemlich fertig. Zum Abschluß wird noch der Befehl JP START (Zeile 36) eingefügt, der eine beliebig häufige Ausführung des Programms durch Rücksprung an den Programmbeginn bewirkt und zeigt, daß auch im System Z80 absolute (auch konditionierte!) Sprünge zu jeder beliebigen Stelle des 64 kByte-Arbeitsspeichers möglich sind.
An dieser Stelle muß noch einiges über Programmiertechnik im Allgemeinen auch unser Beispielprogramm im Besonderen klargestellt werden.

a) Zur Programmierung:

Das gestellte Problem kann mit fast unendlich viel verschiedenen Befehlskombinationen gelöst werden oder anders ausgedrückt, unsere Lösung ist weder die einzig mögliche noch der Stein des Weisen. Wahrscheinlich ist die von Ihnen gefundene Lösung entweder dynamisch oder statisch leistungsfähiger als unsere, d.h. daß sie mit einer kürzeren Verarbeitungszeit oder aber mit geringerem Speicherbedarf auskommt.
Es ist sogar denkbar, daß sie **sowohl** schneller **als auch** mit geringerem Speicherbedarf das gleiche fertigbringt wie das gezeigte ausgeführte Beispiel.

b) Zum Beispielprogramm:

Das Programm nutzt aus didaktischen Gründen bewußt nicht die **volle** Fähigkeit der Z80-CPU aus.

So wurde bereits erwähnt, daß der Sprungbefehl in Zeile 36 ebensogut wie in Zeile 25 und 28 in schnellerer und ökonomischerer Weise als relativer Sprung ausgeführt werden könnte.

Das Löschen des Akkus in Zeile 14 wurde mit einem Konstanten-Ladebefehl erzielt, der 7 Taktzyklen benötigt und zwei Byte im Programmspeicher beansprucht.

Die gleiche Funktion ließe sich durch ein Exclusiv-Oderieren des Akkumulators erreichen, da jede Binärzahl mit sich selbst exclusiv-oderiert als Ergebnis Null bringt.

Der Zugehörige Befehl

XOR A

beansprucht nur 4 Taktzyklen und 1 Byte Programmspeicher. Ebenso lassen sich auch andere Programmfunktionen "trickreicher" durchführen.
Insbesondere soll an dieser Stelle auf Möglichkeiten der Z80-CPU und Notwendigkeiten der Mikrocomputertechnik hingewiesen, die im angeführten Programmbeispiel nicht ausgeführt wurden:

- Der Stackpointer ist grundsätzlich in jedem Programm auf einen sinnvollen Wert im Rahmen der Hardwarekonfiguration des Anwendersystems zu bringen.
  Anderenfalls ist der Stackbereich undefiniert und Interrupt-Serviceroutinen und Unterprogramme können nicht korrekt ausgeführt werden.
- Die Z80-CPU kann
  * mit einem einzigen Befehl Einzelbits in CPU-Registern und im Speicher Setzen, Rücksetzen und Testen, was für Steuerungsaufgaben besonders wichtig ist.
  * 16 bit-Additionen und Subtraktionen durchführen.
  * Schiebebefehle ausführen und Rotationen in Speicherzellen.
  * mit einem einzigen Befehl 64 kByte Speicherinhalte transferieren oder den gleichen Speicherbereich nach einem bestimmten Suchbegriff durcharbeiten.
  * extrem schnell auf Interrupts reagieren, da sie über 2 hardwaremäßige Interrupt-Eingänge und zwei vollständige Arbeitsregistersätze verfügt.
  * Ein/Ausgabe über sämtliche Arbeitsregister (nicht nur den Akku) handeln.
  * 256 Bytes große Blöcke mit einem einzigen Befehl ein- oder ausgeben. Dabei wird der Byte-Zählerinhalt dem Adreßbus verfügbar gemacht, so daß Multiplexaufgaben besonders leicht lösbar werden.
  * BCD-Zählen direkt bearbeiten
  * indiziert adressierende Operationen durchführen
  * Momentanwerte an jeder beliebigen Programmstelle von der CPU in den Arbeitsspeicher auslagern und von diesem in die CPU einlesen.
  * Speicherindirekt - adressierte Interrupt-Behandlung ohne zusätzliche Hardware durchführen.
  * Sämtliche Transferbefehle mit einem einzigen symbolischen Befehl durch sein Assemblerprogramm formulieren lassen.
  * 64 kByte Speicher (auch dynamischer ohne zusätzliche Hardware) adressieren.
  * Kombinationen verschiedener Adreßierweisen bearbeiten.

4.3.2. Beispiel für ein arithmetisches Mikrocomputerprogramm

Bekanntlich verfügen heute handelsübliche 8bit-Mikroprozessoren lediglich über festverdrahtete 8bit-Additionsbefehle, weiterentwickelte Maschinen wie Z80 auch über festverdrahtete 8- und 16bit-Additions- und Subtraktionsbefehle.

Benötigt man in einem solchen Mikrocomputeranwendersystem andere Rechnungsarten, so muß man diese mit Hilfe von Software implementieren. Da die Vorgehensweise dabei für den Anfänger interessant ist und dabei gleich das in Abschnitt 1.8 vermittelte Wissen angewandt werden kann, soll hier eine 8bit-Multiplikationsroutine angegeben und erläutert werden. Den zugehörigen Programmablauf zeigt Bild 67. Man sieht, daß das Rechenverfahren angewandt wurde, das in Pkt. 1.8. besprochen worden war.

Als Register für den 8bit-Multiplikator wurde C gewählt; das Ergebnis kann 16bit lang sein, da die Multiplikation der zwei höchstmöglichen 8stelliger 255 x 255 = 65025 groß werden kann. Aus diesem Grund wurde für Zwischenergebnis und Ergebnis ein 16bit-Registerpaar verwendet (HL). Ebenso mußte für den Multiplikanden ein Registerpaar (DE) reserviert werden, da das Verfahren ja darin besteht, den um eine Potenzstelle nach links verschobenen Multiplikanden sukzessive auf das Zwischenergebnisfeld aufzuaddieren.

War das Carry-Flag = 0, so entfällt die Addition; in beiden Fällen wird jedoch der Inhalt von DE um eine Binärstelle erhöht, d.h. um ein bit nach links geschoben.

Am Ende des Durchlaufs wird geprüft, ob bereits der ganze Multiplikator abgearbeitet ist, d.h. ob das Durchschieben des Multiplikators 8mal stattgefunden hat. Ist der Zähler = 0, so steht im Registerpaar HL das fertige Multiplikationsergebnis, sonst ist der Programmablauf ab der Marke MLOOP zu wiederholen.

Die Codierung (s. Listing Bild 68) ist recht einfach. Bemerkenswert ist dabei die einfache Realisierung eines 16bit-Schiebevorganges mit Hilfe der zwei Befehle SLA r und RL r. Dabei wird das höchstwertige Bit des Registers E durch SLA E ins Carry-Flag geschoben und die unterste Stelle mit einer Null aufgefüllt.

RL D "holt" dann den Übertrag aus dem E-Register über das Carry-Flag ab.

Das ganze Programmstück kann selbstverständlich auch als Unterprogramm geschrieben werden.

In diesem Fall wäre bei Programmbeginn und - Ende ein Retten bzw. Wiederherstellen der Registerinhalte über 4 PUSH und 4 POP-Befehle zu besorgen und die Programmzeilen 18 und 19 durch einen Rücksprungbefehl zu ersetzen.

Eine weitere Alternative wäre die Implementierung dieses Programmes als Makro. Hierzu werden wir noch jetzt ein eigenes Beispiel behandeln. Zuvor soll jedoch noch der Ablauf des eben beschriebenen Multiplikationsprogrammes an einem

Beispiel gezeigt werden, an dem die Funktion der einzelnen Operationen evident wird (Bild 69). Gezeigt werden die Zwischenergebnisse der einzelnen Schleifendurchläufe bei einem Multiplikandenwert = 17 und einem Multiplikator = 11. Das Ergebnis 187 steht in binärer Form bereits ab dem 4. Durchlauf im HL-Reregisterpaar zur Verfügung, da die führende 4 bits des Multiplikators 0 sind und dadurch kein weiterer Beitrag zur Veränderung des Rechenergebnisses geliefert wird (keine Addition auf HL).

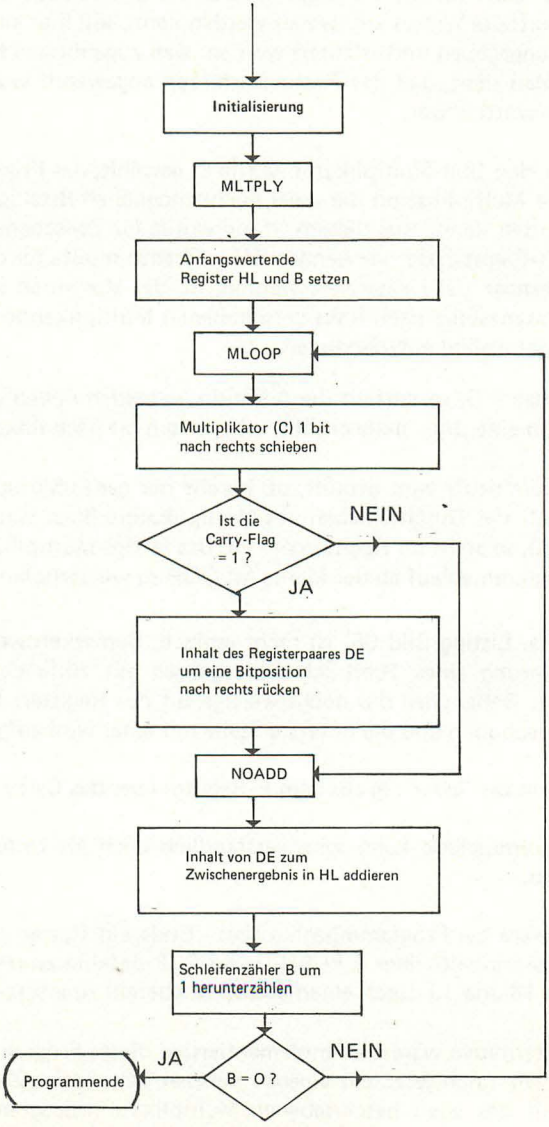


Bild 67: Programmablaufplan für Multiplikation

```
                                  MLTPLY LISTING 2/1/77                PAGE   1
 LOC    OBJ CODE    STMT SOURCE STATEMENT                        ASM VERSION 3.0

                      1            ;8-BIT BINARY MULTIPLY ROUTINE
                      2            ; MULTIPLIER IN C
                      3            ; MULTIPLICAND IN DE
                      4            ; PARTIAL SUM IN HL
0000                  5           ORG   0
0000    210000        6 MLTPLY:   LD    HL,0
0003    0608          7           LD    B,8
                      8
0005    CB39          9 MLOOP:    SRL   C
0007    3001         10           JR    NC,NOADD
0009    19           11           ADD   HL,DE
                     12
000A    CB23         13 NOADD:    SLA   E
000C    CB12         14           RL    D
                     15
000E    10F5         16           DJNZ  MLOOP
                     17
0010    76           18           HALT
                     19           END

   0 ASSEMBLY ERRORS
```

Bild 68: Listing des Multiplikationsprogramms

4.3.3. Beispiel eines Makroprogrammes

Im folgenden Programmbeispiel (Bild 70) ist ein Makro zum Transfer eines n Byte langen Blocks von einer bestimmten Sonde - zu einer frei vorgebbaren Zieladresse angegeben.

Da der Programmablauf praktisch trivial ist, wurde hier auf einen Programmablauf verzichtet.

Er besteht im wesentlichen aus einem einzigen Befehl, (Zeile 16), nämlich dem zum Datentransfer. Der Rest dient lediglich zum Retten der Register (Zeilen 9 - 11 und 18 - 20).

Dieses Retten kann über die bedingte Assemblierung (Zeilen 8 und 12 und 17 und 21) ausgeschaltet werden (s. Z80-Programmierhandbuch). Der Sinn dieser bedingten Assemblierung ist, das gleiche Programm (= hier den gleichen Makro) in unterschiedlicher Funktion zu verwenden (hier: einmal mit und einmal ohne Retten der Register, je nachdem, ob das bei jeweiliger Verwendung des Makros nötig ist oder nicht).

Durchlauf	Inhalt von Register am Ende des jeweiligen Durchlaufs			
	B	C	E	HL
0 (Programm-beginn)	8	00001011	0010001	0
1	7	00000101	00100010	00000000 00010001 + 00010001
2	6	00000010	01000100	00010001 00100010 + 00110011
3	5	00000001	10001000	00110011
4	4	0	00010000	00110011 10001000 + 10111011
5	3	0	00100000	10111011
6	2	0	01000000	10111011
7	1	0	10000000	10111011
8	0	0	0	10111011

Bild 69: Ablaufbeispiel

		D	E
Multiplikant	= 17 =	00000000	00010001
			C
Mulitpilkator	= 11 =	00000000	00001011
		H	L
Ergebnis	= 187 =	00000000	10111011

```
                                ASMTST LISTING 2/1/77                    PAGE   2
 LOC    OBJ CODE    STMT SOURCE STATEMENT                          ASM VERSION 3.0

                     28 *EJECT
                     29 ;
                     30 ;HERE IS A SIMILAR USE WHICH DOESN'T SAVE REGS
                     31 ;
                     32         MOVE  OUTBUF INBUF 25
                                    COND  .NOT.(''='')
                                    PUSH  BC
                                    PUSH  DE
                                    PUSH  HL
                                    ENDC
0011    011900                      LD    BC,25  ;MOVE 25 BYTES
0014    111C00                      LD    DE,OUTBUF
0017    213500                      LD    HL,INBUF
001A    EDB0                        LDIR
                                    COND  .NOT.(''='')
                                    POP   HL
                                    POP   DE
                                    POP   BC
                                    ENDC
                     33
001C                 34 OUTBUF: DEFS  25
0035                 35 INBUF:  DEFS  25
                     36         END

   0 ASSEMBLY ERRORS

PASS 2 COMPLETE



CROSS REFERENCE                 ASMTST LISTING 2/1/77                    PAGE   3
SYMBOL  VAL  DEFN REFS

INBUF   0035   35   27   32
MOVE    0000    7   27   32
OUTBUF  001C   34   27   32

ASSEMBLY COMPLETE
OS>
```

Fortsetzung nächste Seite

```
OS>A ASMTST.S - (P M X D=2/1/77)
PASS 1 COMPLETE


                                ASMTST LISTING 2/1/77                    PAGE  1
 LOC    OBJ CODE     STMT SOURCE STATEMENT                          ASM VERSION 3.0

                        1 ;
                        2 ; MOVE MACRO--BLOCK MOVE OF #LENGTH BYTES FROM
                        3 ;              #SOURCE TO #DEST
                        4 ;              IF 4TH PARAMETER IS PRESENT, THEN
                        5 ;              REGISTERS ARE PRESERVED
                        6 ;
                        7 MOVE      MACRO  #DEST #SOURCE #LENGTH #SAVE?
                        8           COND  .NOT.('#SAVE?'='')
                        9           PUSH  BC
                       10           PUSH  DE
                       11           PUSH  HL
                       12           ENDC
                       13           LD    BC,#LENGTH  ;MOVE #LENGTH BYTES
                       14           LD    DE,#DEST
                       15           LD    HL,#SOURCE
                       16           LDIR
                       17           COND  .NOT.('#SAVE?'='')
                       18           POP   HL
                       19           POP   DE
                       20           POP   BC
                       21           ENDC
                       22           ENDM
                       23
                       24 ;HERE IS A SAMPLE USE OF 'MOVE' WHICH SAVES THE
                       25 ;REGISTER CONTENTS
                       26 ;
                       27      MOVE  OUTBUF INBUF 25,S
                                     COND  .NOT.('S'='')
0000    C5                           PUSH  BC
0001    D5                           PUSH  DE
0002    E5                           PUSH  HL
                                     ENDC
0003    011900                       LD    BC,25  ;MOVE 25 BYTES
0006    111C00                       LD    DE,OUTBUF
0009    213500                       LD    HL,INBUF
000C    EDB0                         LDIR
                                     COND  .NOT.('S'='')
000E    E1                           POP   HL
000F    D1                           POP   DE
0010    C1                           POP   BC
                                     ENDC
```

Bild 70: Beispiel eines Makroprogramms

4.3.4. Prüfprogramm für den Z80-KIT

Als weiteres Beispiel soll hier ein Programm gezeigt werden. das die Funktionsfähäufigkeit des in Punkt 3.2. beschriebenen KIT testet. Seinen Ablauf zeigt Bild 71

Das Programm besteht aus zwei ineinander verschachtelten Zählerschleifen, an deren Enden eine Ausgaberoutine für eines der Hexadezimalen Anzeigeelemente steht. Ausgegeben wird der Inhalt von Register DE, der nach jedem Durchgang decrementiert wird.

Beginnend mit F werden so alle hexadezimalen Elemente in absteigender Folge nacheinander angezeigt. Nach 0 folgt wieder F, so daß sich eine unendliche Folge ergibt.

Wichtigster Teil des Testprogramms ist die Ausgaberoutine, die es ermöglicht, bestimmte Werte auf der Anzeige darzustellen. Diese Routine , die auch vom Monitorprogramm benutzt wird, basiert auf folgendem Gedanken:

Ein anzuzeigender Wert (hexadezimal!) wird in das DE-Registerpaar eingelesen, wobei das D-Register stets nur 0 enthalten darf. Der Inhalt des DE-Registerpaares wird nun zum HL Registerpaar addiert (deshalb war es notwendig, das DE-Registerpaar und nicht nur das E-Register zu verwenden), welches die Anfangsadresse einer Tabelle (SEGM) enthält. Nach derAddition enthält das HL Registerpaar eine neue Adresse und zwar den Wert:

Anfangsadresse + Inhalt DE-Registerpaar

Unter dieser Adresse ist der Wert abgespeichert, der bei Ausgabe auf die Anzeigeeinheit den im DE Register abgespeicherten Wert darstellt.

z.B.: Ist in DE der Wert 8 gespeichert, so ergibt sich:

		D	E
<DE>	=	00	08

Im HL Registerpaar steht die Anfangsadresse der Tabelle SEGM:

		H	L
<HL>	=	3 C	7 2

Nach der Addition (hexadezimal) ergibt sich:

		H	L
<HL>	=	3 C	7 A

Unter dieser Adresse findet man den Wert

<3C 7A> = 7 F
adr wert

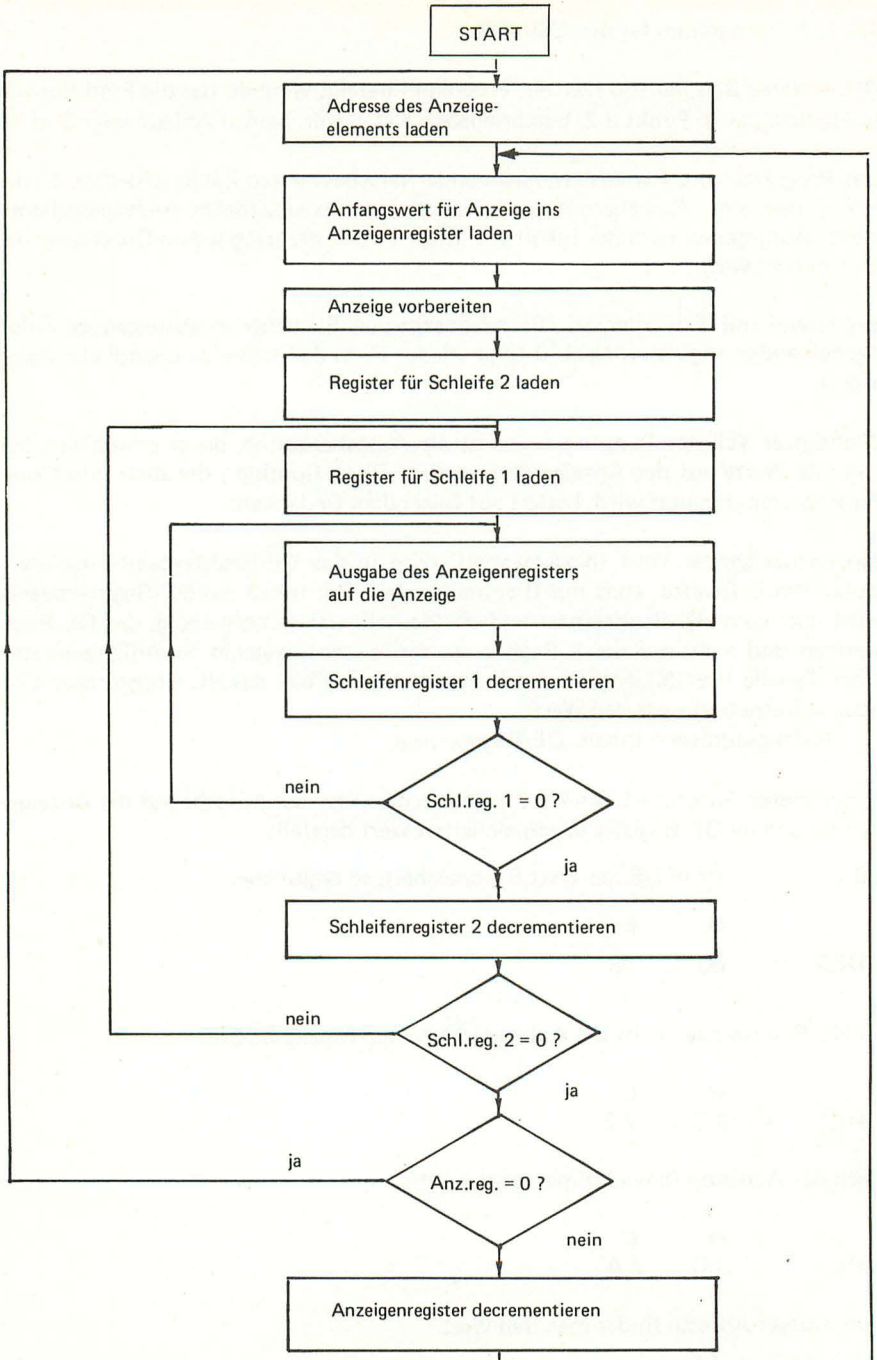


Bild 71: Ablaufplan eines Prüfprogramms

Bei Ausgabe dieses Wertes zeigt das Anzeigeelement den Wert 8.

Die Ansteuerung der einzelnen Anzeigeelemente geschieht mit Hilfe des Registers C und einer zugehörigen Port-Tabelle, die dem Listing des KIT-Programms entnommen werden kann. Soll die Ausgabe z.B. auf das obere Datenanzeigeelement erfolgen, ist C nicht mit 0C sondern mit 0D zu laden.

Die Zeitkonstanten sind jeweils im B Register untergebracht. Nach dem erstmaligen Laden mit dem Wert für Schleife 2 (00F), wird der Inhalt des B Registers in den Stack ausgelagert (und damit "konserviert"). B wird erneut geladen und zwar mit dem Wert für Schleife 1 (0FF). Wird Wert 1 wieder gebraucht, so wird er aus dem Stack geholt.

Durch Veränderung der beiden Werte kann ein verschieden schneller Durchlauf der Anzeige erreicht werden.
z.B.: Lädt man statt 0FF den Wert 00F, so ergibt sich ein wesentlich schnellerer Durchlauf.

```
                                TEST   LISTING                          PAGE   1
 LOC   OBJ CODE   STMT SOURCE STATEMENT                                 ASM 3.0

3C50                  1  ORG 3C50H
3C50   0E0C           2  LD C,0CH;          ADRESSE DES ANZ.ELEMENTES LADEN
3C52   110F00         3 NEU:LD DE 000FH;    ANZ.REG. MIT ANFANGSWERT LADEN
3C55   21723C         4 START:LD HL,SEGM;   ANFANGSADRESSE TABELLE LADEN
3C58   19             5  ADD HL,DE;         TABELLENADRESSE GENERIERT
3C59   7E             6  LD A,(HL);         INHALT TABELLE GELADEN
3C5A   060F           7  LD B,00FH;         SCHLEIFE 2 LADEN
3C5C   C5             8 S2:PUSH BC;         WERT IN DEN STAPELSPEICHER ABLEGEN
3C5D   06FF           9  LD B,0FFH;         SCHLEIFE 1 LADEN
3C5F   00            10  S1:NOP
3C60   ED79          11  OUT (C),A;         AUSGABE
3C62   10FB          12  DJNZ S1;           SCHL.1 DECR.,WENN NICHT 0,SPR.NACH S1
3C64   00            13  NOP
3C65   C1            14  POP BC;            WERT VON SCHL.2 AUS DEM STPL.SP.HOLEN
3C66   10F4          15  DJNZ S2;           SCHL.2 DECR.,WENN NICHT 0,SPR.NACH S2
3C68   3E00          16  LD A,0
3C6A   83            17  ADD A,E
3C6B   CA523C        18  JP Z,NEU;          WENN ANZEIGE 0,DANN SPRUNG NACHRT IN D
3C6E   1D            19  DEC E;             ANZ.REG. DECREMENTIEREN
3C6F   C3553C        20  JP START;          DAS GANZE VON VORNE MIT NEUER ZAHL
3C72   5F            21  SEGM:DEFB 05FH;    ANFANGSWERT TABELLE
3C73   06            22  DEFB 06H
3C74   3B            23  DEFB 3BH
3C75   2F            24  DEFB 2FH
3C76   66            25  DEFB 66H
3C77   6D            26  DEFB 6DH
3C78   7D            27  DEFB 7DH
3C79   07            28  DEFB 07H
3C7A   7F            29  DEFB 7FH
3C7B   6F            30  DEFB 6FH
3C7C   77            31  DEFB 77H
3C7D   7C            32  DEFB 7CH
3C7E   59            33  DEFB 59H
3C7F   3E            34  DEFB 3EH
3C80   79            35  DEFB 79H
3C81   71            36  DEFB 71H
                     37 END

   0 ASSEMBLY ERRORS
```

Bild 72: Listing des Prüfprogramms

4.3.5. Praktische Beispiele einer Programmentwicklung

Zur Konkretisierung des Entwicklungsablaufs nach Bild 57 und 59 soll hier dieser Ablauf auszugsweise an einem Beispiel Schritt für Schritt durchgesprochen werden. Das Anwendungsbeispiel sei ein einfaches Reaktionstest-Spiel mit dem MP Z80-CPU.
Wir wollen annehmen, daß die Problemdefinition ergeben hat:

- Spielbeginn: Nach Drücken einer Starttaste leuchtet nach einer vom Zufall bestimmten Zeit irgendeine der Ziffern 0 ... 9 auf. (Mikrocomputer-Vorgabe)
- Der Spieler muß dann eine Taste mit der richtigen Ziffer in einem Zehnertastenfeld drücken.
- Die Zeit bis zum Drücken der richtigen Ziffer wird gemessen nach 2 sec. Abbruch des Spiels.
- Drückt er eine falsche Taste, wird dies ignoriert.
- Drückt er mehr als eine Taste, wird das Spiel abgebrochen.
- Ist die richtige Taste gedrückt, wird die Reaktionszeit abgespeichert und angezeigt.

Danach erfolgt eine Mitteilung der Reaktionszeit und Vergleich mit den Vor-Spielern; durch Drücken der Start-Taste ist erneuter Spielbeginn möglich.

- Datenflußplan nach Bild 73
- Nötige Mikrocomputer-Bausteine:

  1 Zentralbaustein Z80-CPU
  1 Taktgenerator-Baustein (1/2 Standardgatterbaustein SN 74 LS04)
  2 RAM's (statisch, 1/4 kByte) z.B. 8211 - 1
  2 PROM's (1/2 kByte) HM 7641
  1 Ein/Ausgabebaustein Z80-PIO

- Geschätzter Speicherbedarf:

  50 Bytes Lese/Schreibspeicher (RAM)
  500 Bytes Festwertspeicher (PROM)

Damit läßt sich das gesamte Konzept mit 1 Stück Z80-ECB/C-Computerplatine oder mit der Grundausstattung des Z80-KIT realisieren.

- Programmumfang:

  500 Bytes

- Software-Entwicklungsaufwand (incl. Sicherheitsfaktor):

* detaillierter Programmablaufplan:	2 Tage
* Codierung	1 Tag
* Hardware-Aufbau und -Test	0,5 Tage
* Eingabe, Übersetzung, Test:	1,5 Tage
	5 Tage = 1 Woche

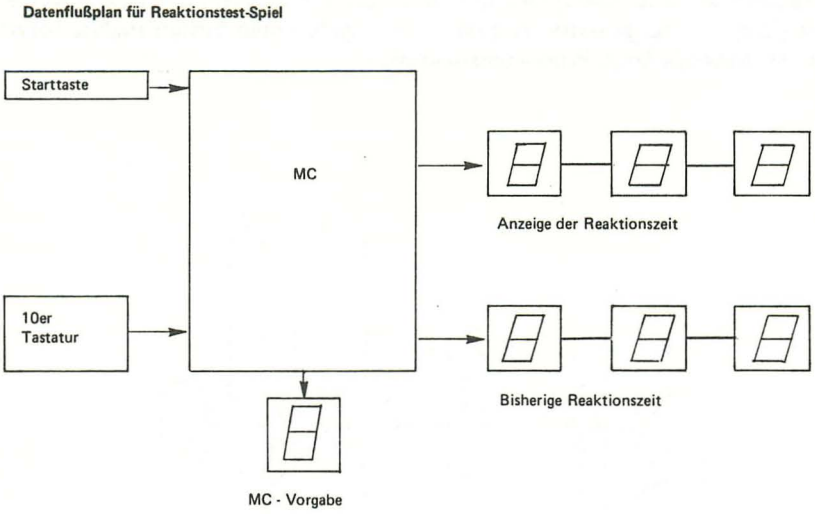
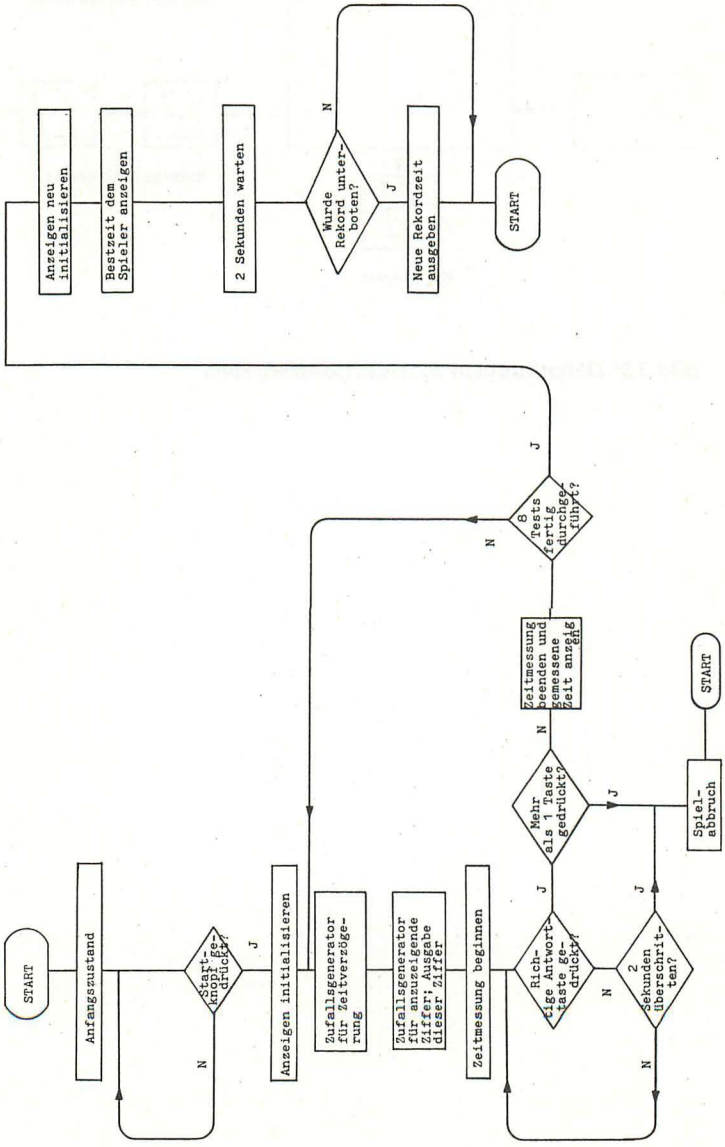


Bild 73: Datenflußplan für Reaktionszeit-Spiel

Daraufhin sei eine Kostenrechnung durchgeführt worden, die eine positive Entscheidung zur Folge hatte. Aus der dann angefertigten Systemanalyse resultierte u.a. der folgende Grob-Programmablaufplan

Bild 74: GROB-PROGRAMMABLAUFPLAN



Als Hardware wird der 1-Platinen Mikrocomputer im Europaformat Z80-ECB verwendet; an Hardware-Entwicklung ist also lediglich der Anschluß des ebenfalls fertig zu beziehenden Tastenfeldes und der Anzeigen durchzuführen.

Gleichzeitig mit Erstellung dieser Schaltungen, die sich sämtlich auf der Gerätefrontplatine befinden, beginnt die Programmentwicklung.
Sie führt zunächst über die Grundlage des Flußdiagrammes im Bild 74 zu einem detaillierten Programmablaufplan. Da der Programmumfang nicht groß ist, entschied man sich zur Programmierung in ASSEMBLER-Sprache.

Der Software-Entwickler benützt nun das Mikrocomputer-Entwicklungssystem, ruft das EDITOR-Programm auf und tippt seinen Primärprogrammtext ein. Eine Datei mit dem Namen SPIEL und der Dateiart S wird aufgebaut und abgespeichert, d.h. dm EDITOR wird zu Beginn des Editiervorgangs "gesagt", daß dem folgenden Text der Dateiname SPIEL zuzuordnen ist. Ein Beispiel für einen Editiervorgang zeigt Bild 75, dem das Beispielprogramm von Pkt. 4.3.2. zugrunde liegt. Nun lädt der Entwickler in den Arbeitsspeicher des Mikrocomputer-Entwicklungssystems anstelle des EDITORS das ASSEMBLER-Programm und teilt dem Mikrocomputersystem mit, daß das unter dem Namen SPIEL abgelegte Programm zu übersetzen ist. Daraufhin erzeugt das Mikrocomputer-Entwicklungssystem ein Objektprogramm wieder unter dem Namen SPIEL, das jedoch vom ASSEMBLER automatisch durch eine andere Dateiartkennung (U für User-Code) markiert wurde und dadurch vom Primärprogramm SPIEL unterscheidbar ist.

Zu Testzwecken ruft der Entwickler das Objektprogramm nun auf und läßt es ablaufen. Sobald evtl. Fehler beseitigt sind und der Z80-ECB mit den übrigen Hardware-Komponenten (Tastatur, Anzeigen) beschaltet ist, wird die CPU der Platine aus seiner Fassung herausgezogen und durch den Testadapter-Stecker ersetzt. Der Entwickler prüft nun das komplette System (= Spiel) mit dem Mikrocomputer-Entwicklungssystem und dem Echtzeit-Testadapter auf seine richtige Funktion, lokalisiert und beseitigt evtl. Fehler.

```
OS>E  (S) MLTPLY S
NEW FILE
INPUT
>       ; 8-BIT Binary Multiply Routine
>       ; Multiplier in C
>       ; Multiplicand in DE
>       ; Partial Sum In @@@in HL
>
>mltply  LD HL,ZERO
>        LD B,EIGHT
>
>MLOOP   SRL C
>
EDIT
>T
T>C /IT/it
      ; 8-Bit Binary Multiply Routine
>c /mltply/MLTPLY/* *
MLTPLY  LD HL,ZERO
>u /um
      ; Partial Sum in HL
>r       ; HL contains the partial sum
>t
T>l /HL
      ; HL contains the partial sum
>a
MLTPLY  LD HL,ZERO
>u 4
      ; Multiplier in C
>p 3
      ; Multiplier in C
      ; Multiplicand in DE
      ; HL contains the partial sum
>b
MLOOP   SRL C
>u /in
      ; HL contains the partial sum
>a
      ; Multiplicand in DE
>de /ZERO
>p
MLTPLY  LD HL,ZERO
>LI
0003
>W
0001
0006
>G 2
        Multiplier in C
>PU 3 MLTPLY F S
>SAVE
>P

>N
MLOOP   SRL C
>QUIT
OS>F
```

```
>P MLTPLY F
      ; Multiplier in C
MLTPLY  LD HL,ZERO
        LD B,EIGHT
>Q
OS>E MLTPLY S
EDIT
>B
MLOOP   SRL C
>GE MLTPLY F
        LD B,EIGHT
>U /SRL
MLOOP   SRL C
>R
INPUT
>MLL!MLOOP  SRL D
>       EX DE,HL
>
EDIT
>T
T>P *
      ; 8-Bit Binary Multiply Routine
      ; Multiplier in C
MLTPLY  LD HL,ZERO
        LD B,EIGHT

MLOOP  SRL D
       EX DE,HL
       ; Multiplier in C
MLTPLY  LD HL,ZERO
        LD B,EIGHT
EOF
>M L /EIGHT&C /EIGHT/NINE
>T
T>X
        LD B,EIGHT
        LD B,NINE
>N #00
MLOOP  SRL D
>I
INPUT
>       SRL A
>       ADD HL,BC
>
EDIT
>FILE
OS>
```

Bild 75: Beispiel eines Editiervorgangs

4.4. Programmierung des Mikrocomputer-Programmspeichers

Wir sind mit der Entwicklung unseres Anwendersystems nun so weit, daß wir mit an Sicherheit grenzender Wahrscheinlichkeit sagen können, daß es keine Fehler mehr enthält. Wir können nun an die Programmierung des Mikrocomputer-Programmspeichers denken.

Bekanntlich verwendet man als Programmspeicher
i.d.R. Festwertspeicher. Je nach geplanter Stückzahl können maskenprogrammierte ROM's (bei großen Stückzahlen), die vom Halbleiterhersteller programmiert werden müssen oder PROM's (vom Anwender programmierbare Speicherbausteine) eingesetzt werden. Zur Programmierung dieser PROMs ist ein PROM-Programmiergerät nötig, mit dem der Entwickler ohne Weitergabe seines Programms an Dritte und ohne Zeitverlust PROM-Festwertspeicher einfach durch Eingabe bestimmter Kommandos am Mikrocomputer-Entwicklungssystem selbst programmieren kann.
Die fertig programmierten PROM's (oder aber die vom Halbleiterhersteller bestellten maskenprogrammierten ROM's) können nun in die Mikrocomputer-Anwender-Schaltung eingesetzt werden, die jetzt arbeitsbereit ist.

5. Mikrocomputer-Entwicklungssystem

Wir haben gesehen, daß die Verwendung eines Mikrocomputer-Entwicklungssystems die rationellste Methode zur Entwicklung eines Mikrocomputer-Anwendersystems darstellt und wollen nun zunächst seine einzelnen Hardware-Komponenten und ihre Kombination miteinander betrachten.

5.1. Mindestausstattung

Zweck eines Mikrocomputer-Entwicklungssystems ist, die Arbeit des Entwicklers zu erleichtern, effektiver zu gestalten und zu rationalisieren. Hierfür muß ein Entwicklungssystem mindestens folgende Funktionen ausführen können:

Auszuführende Arbeit		Technische Funktion	
a)	Einlesen, speicher und korrigieren eines Programmtextes mit Computerunterstützung (Editieren)	a1)	Einlesen des Editorprogramms von einem Datenträger
		a2)	Programmtext-Eingabe von einer Tastatur mit Anzeige
		a3)	Ausgabe des Programmtextes auf einen maschinenlesbaren Datenträger
b)	Übersetzen des Primärprogramms in Maschinensprache und abspeichern des Objekt-Codes	b1)	Einlesen des Übersetzungsprogramms von einem Datenträger
		b2)	Einlesen des Primärprogramms
		b3)	Ausgabe des Objektcodes auf einem maschinenlesbaren Datenträger

Diese Funktionen werden von einer Minimalkonfiguration erfüllt, wie sie Bild 76 zeigt.

Sie besteht aus dem Zentralgerät des Z80-Mikrocomputer-Entwicklungssystems, einem Z80-Floppy-Disk-Doppellaufwerk * und einem Datensichtgerät mit Eingabetastatur als Systemkonsole.
Dabei ist im **Zentralgerät** die "Intelligenz" des Systems konzentriert.
Auf der ersten **Floppy-Disk** sind die Programme abgespeichert, die zum Einlesen, Manipulieren und Übersetzen des Anwenderprimärprogrammes dienen. Da diese Programme unter dem Namen Systemsoftware zusammengefaßt sind, heißt die entsprechende Floppy-Disk "System-Platte".
Soll mit einem dieser Programme gearbeitet werden, d.h. z.B. ein Anwenderprimärprogramm erstellt werden, muß das zugehörige Systemprogramm (in diesem Fall das Editorprogramm) erst von der Floppy-Disk in den Arbeitsspeicher (das ist der mit RAM's aufgebaute 16 ... 60 kByte großer Speicher im Zentralgerät)einge-

* Floppy Disks sind preiswerte periphere Speichergeräte, bei denen die Daten auf magnetisierten, biegsamen Platten (daher der Name "Floppy"-Disk) gespeichert sind. Es ist wahlfreier Zugriff möglich, d.h. im Gegensatz zu Lochstreifen oder Bändern kann auf eine beliebige Stelle zugegriffen werden.

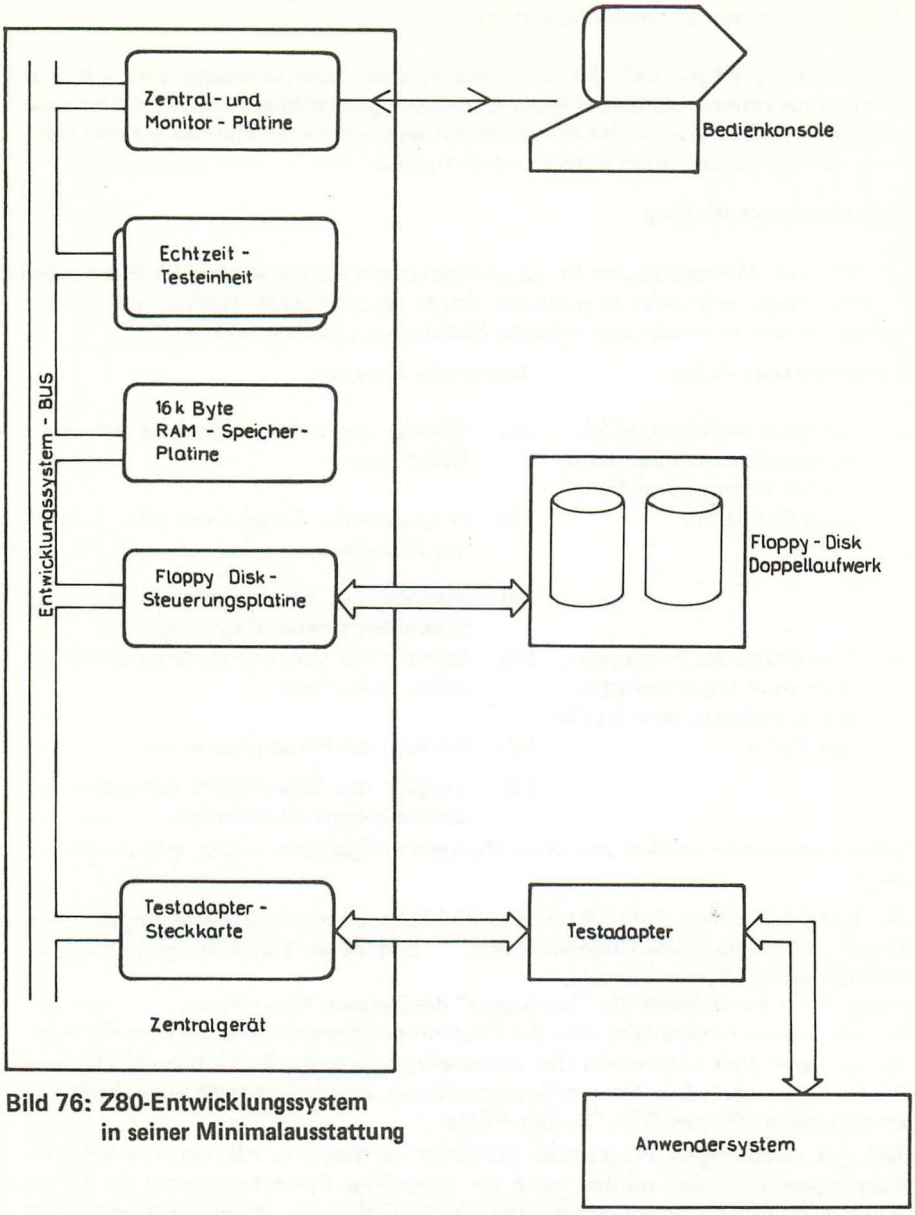


Bild 76: Z80-Entwicklungssystem in seiner Minimalausstattung

** Datensichtgerät oder Fernschreiber (z.B. TTY), das für den Dialog zwischen Bediener und dem Computer reserviert ist.

Sobald das Editorprogramm von der Floppy Disk in den Arbeitsspeicher eingelesen ist, ist das Mikrocomputer-Entwicklungssystem bereit, die Eingabe des Anwenderprogrammtextes zu empfangen.

Diese Eingabe geschieht wieder von der Systemkonsole aus. Ist das Eintippen des gesamten Programmtextes geschehen, muß er "konserviert" werden, denn man will ja in den meisten Fällen auch noch später auf dieses Programm zurückgreifen, beispielsweise, um nach den Tests Korrekturen durchzuführen; der Arbeitsspeicher wird aber in der Zwischenzeit für andere Dinge, beispielsweise die Programmübersetzung, benötigt.

Daher lagert man den fertigen Programmtext auf ein nichtflüchtiges Speichermedium, eben die zweite Floppy-Disk, aus; der Programmtext stellt jetzt eine auf Diskette (so nennt man den Floppy Disk-Datenträger) abgelegte Datei dar. Damit der Programmtext später wieder auffindbar ist, muß man ihn mit einem Dateinamen und einer Dateiart angeben, was zu Beginn der Programmtexteingabe von der Systemkonsole aus vom Benutzer festgelegt wird. Da auf der zweiten Floppy Disk hauptsächlich Dateien des Anwenders gespeichert werden, nennt man sie "Anwender-Diskette".

Wir sehen schon, daß sich alle diese Funktionen mit unserer Minimalkonfiguration durchführen lassen. Entsprechendes gilt für den Übersetzungsvorgang: Hier wird, wie wir in Punkt 4.2. im Prinzip gesehen hatten, zunächst das Übersetzungsprogramm von der Systemplatte aufgerufen, dann das zu übersetzende Primärprogramm von der Anwenderplatte eingelesen, übersetzt und schließlich das fertig übersetzte Anwenderprogramm - das Objektprogramm - ebenfalls auf der Anwenderplatte "konserviert".

Ebenso ist ein Testen des Anwenderprogramms durch Aufruf des residenten Fehlersuchprogrammpakets und Korrektur des fehlerhaften Anwenderprogramms durch erneute Verwendung des Editorprogramms möglich.

Es fehlt nun noch ein zweiter Schritt, um das Anwendersystem betriebsbereit zu machen: Das Abspeichern des Anwenderprogramms in Festwertspeichern und der Systemtest incl. evtl. Korrekturen.

Dieser Vorgang kann bei Verwendung der Minimalkonfiguration nur folgendermaßen geschehen.

- Erstellung des fertig programmierten Festwertspeichers durch den Halbleiterhersteller oder seines Distributors aufgrund des auf Diskette gespeicherten Objektprogramms und
- Durchführung des Systemtests durch Ablaufenlassen des in PROM's gespeicherten Anwenderprogramms auf der fertigen Anwenderhardware.

Da beide Aktionen recht umständlich und vor allem zeitaufwendig sind, wird man nach Möglichkeit nicht mit der Minimalkonfiguration, sondern mit einer erweiterten Version arbeiten, die anschließend beschrieben wird.

5.2. Erweiterungsmöglichkeiten

Die in Bild 77 gezeigte erweiterte Konfiguration besteht aus der Minimalkonfiguration und zusätzlichen Einrichtungen, die je nach den gestellten Anforderungen an den Komfort des Systems einzeln oder alle gleichzeitig hinzugefügt werden können. Zusätzlich ist der Anschluß von beliebigen anderen Geräten, wie Lochstreifenlesern/stanzern, Magnetbandgeräten und sogar Direktanschluß an andere Rechner (z.B. Großanlagen über ein Modem und eine Wahlleitung) möglich; da dies jedoch Sonderfälle bleiben werden, sollen sie in dieser grundlegenden Broschüre nicht behandelt werden.

Zu betonen ist jedoch, daß man statt der Datensichtstation selbstverständlich auch Fernschreiber mit RS 232 oder Strompräge-Serienschnittstelle (z.B. Teletype ASR 33) anschließen kann, wodurch man Ausdrucke und Lochstreifenerzeugung sozusagen "geschenkt" bekommt, aber gleichzeitig den Nachteil der geringeren Zuverlässigkeit und Lärmbelästigung solcher Geräte in Kauf nehmen muß.

Die erste, am leichtesten verständliche Zusatzoption gegenüber der Minimalkonfiguration ist der Anschluß eines **Schnelldruckers.** Dadurch ist die zeitsparende Ausgabe von Protokollen (beispielsweise Programm-Listing) möglich, was für Dokumentationszwecke wichtig ist.

Soll die in Pkt. 4.2 beschriebene Schwierigkeit des Systemtests durch Ablauf des Anwenderprogramms vermieden werden, ist der **Echtzeit-Testadapter** zu verwenden; mit ihm werden Mikrocomputer-Entwicklungssysteme und Anwenderhardware elektrisch miteinander verbunden und der Systemtest in Echtzeit durchgeführt.
Dabei braucht das Anwenderprogramm noch nicht in einem Festwertspeicher abgelegt zu sein, d.h. man braucht seine PROM's erst dann zu programmieren oder programmieren zu lassen, wenn sicher ist, daß das gesamte Anwendersystem (Anwenderhardware + Anwendersoftware) fehlerfrei miteinander arbeiten und spart so Zeit und PROM-Programmierkosten. Der Echtzeittestadapter besteht aus einer Steckkarte, die in das Zentralgerät eingeschoben wird, einem Stecker, der anstelle der Anwender CPU in die Anwendercomputerschaltung eingesteckt wird und einem Kabel, das diese beiden Komponenten miteinander verbindet.

Soll auch die letzte Problematik der Minimalkonfiguration, nämlich die Notwendigkeit für den Anwender, sich PROM's "außer Haus" programmieren zu lassen, ausgeräumt werden, muß ein **PROM-Programmiergerät** hinzugefügt werden. Solche Geräte sind als externe, selbständige Einheiten verfügbar, zusätzlich wird eine Billigversion angeboten, die direkt in das Entwicklungssystem eingesteckt wird.

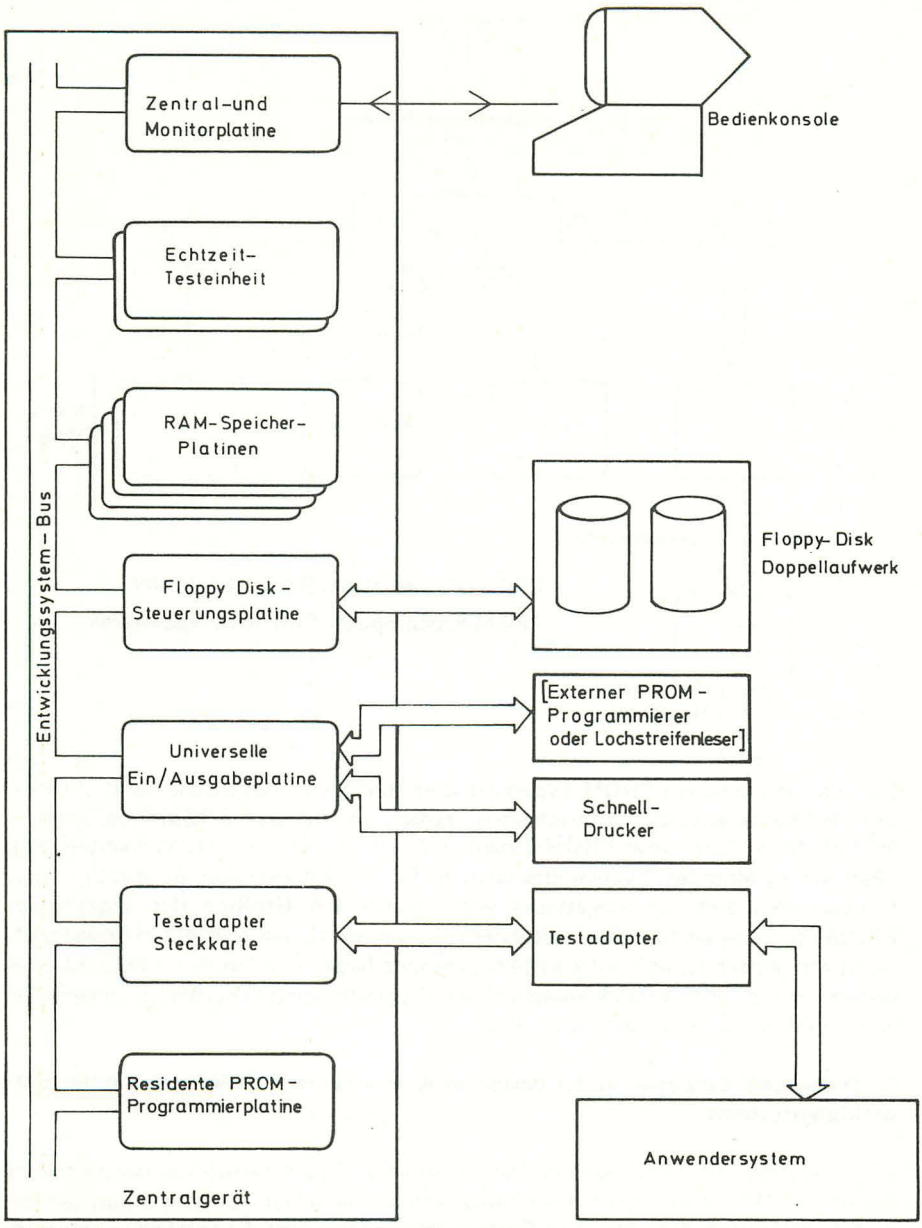


Bild 77: Entwicklungssystem mit Erweiterungen

Neben dem Einschreiben von Daten aus dem Arbeitsspeicher des Entwicklungscomputers bzw. von einem maschinenlesbaren Datenträger in ein PROM, ist mit diesen Geräten auch das Auslesen, Vergleichen und Kopieren von PROM-Inhalten möglich.

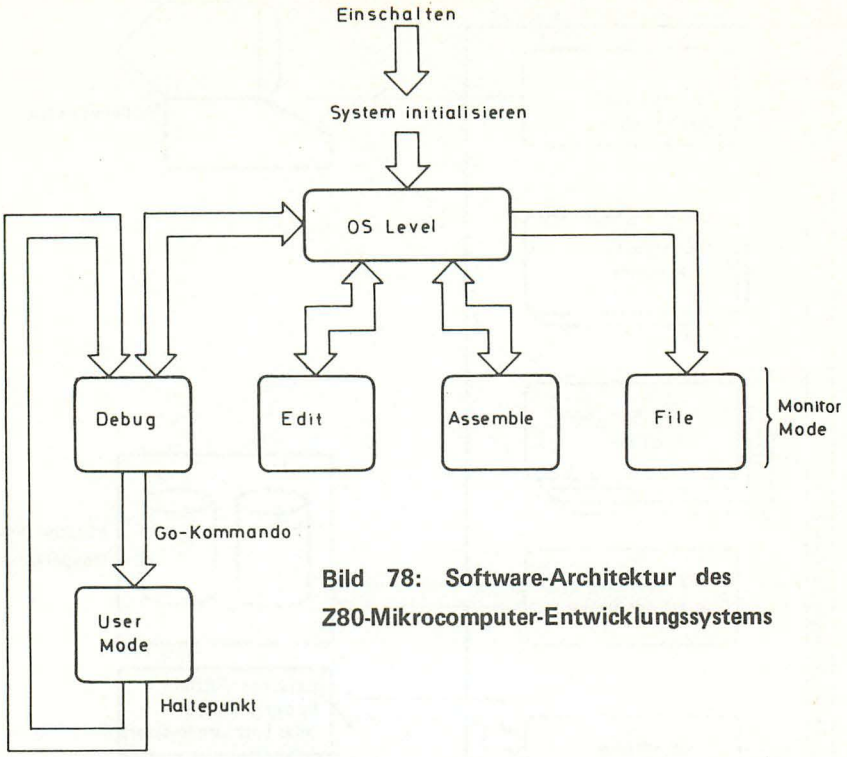


Bild 78: Software-Architektur des Z80-Mikrocomputer-Entwicklungssystems

Da sich die einzelnen PROM-Typen oft stark durch ihre elektrischen und physikalischen Charakteristiken unterscheiden, müssen im Universal-PROM-Programmiergeräten für verschiedene PROM-Typen unterschiedliche Anpassungsmodule (sog. "Personality-Modules") verwendet werden. Bei der Billigversion, die direkt in den Entwicklungscomputer eingesteckt wird, wurde ein Großteil der allgemeinen PROM-Programmierfunktion softwaremäßig gelöst, so daß sich der Hardwareaufwand fast ausschließlich auf bausteintypenspezifische Elektronik beschränkt. Aus diesem Grund gibt es daher ausschließlich Programmiersteckkarten, die jeweils für eine PROM-Familie verwendet werden.

6. Technische Einzelheiten der einzelnen Komponenten des Mikrocomputer-Entwicklungssystems.

Bisher wurden die Aufgaben und das prinzipielle Zusammenwirken der einzelnen Geräte im Mikrocomputer-Entwicklungssystem klargelegt. Es sollen nun technische Einzelheiten der einzelnen Geräte erläutert werden; die folgenden Informationen sind wegen der Verschiedenartigkeit der lieferbaren Produkte zwangsläufigerweise produktspezifisch.

Wie bisher, wollen wir dabei hauptsächlich Komponenten des Z80-Systems betrachten. Eine Zusammenfassung der technischen Daten des Z80-Mikrocomputer-Entwicklungssystems finden Sie in Anhang 2.

6.1. Das Zentralgerät des Z80-Mikrocomputer-Entwicklungssystems

Kernstück des Entwicklungssystems ist der Mikroprozessor Z80-CPU. Durch seine Multi-Task-fähige Architektur ist er besonders gut für das Mikrocomputer-Entwicklungssystem geeignet, der sowohl in dem residenten Monitor (Monitor-Mode) als auch mit dem Anwendersystem (users mode) operieren kann.

In der Monitor-Betriebsart (Monitor-Mode) arbeitet das Gerät als autonome Entwicklungshilfe, wobei Anwender-Programm in den Schreib-/Lesespeicher des Systems eingegeben, editiert, übersetzt, korrigiert, auf Diskette abgespeichert, von der Diskette geladen und zum Ablauf gebracht werden können.
All diese Funktionen werden mit einfach zu erlernenden Kommandos von der Bedienkonsole aus angestoßen.
Im Users Mode werden Speicher und Peripherie des Entwicklungssystems einer fertigen Anwenderschaltung verfügbar gemacht, wobei ein ins Entwicklungssystem-RAM eingelesenes Anwenderprogramm unter Echtzeitbedingungen ausgeführt wird.
Mit dem Hardware-Adapter ("Users Interface") kann im Test auch Speicher oder Peripherie des Anwender-Systems mit den Funktionseinheiten des Entwicklungssystems kombiniert werden.
Unabhängig von den Möglichkeiten des Echtzeittestadapters verfügt das Zentralgerät des Z80-Entwicklungssystems standardmäßig über eine besonders attraktive Einrichtung, seine Echtzeit-Testeinheit, mit der sich fragliche Anwendersystemzustände unter Echtzeitbedingungen in einen hierfür reservierten zusätzlichen Speicher ablegen und anzeigen oder aber als Rücksprungursache in den Monitor-Mode verwenden lassen.

Seine **Zentraleinheit- und Monitor-Steckkarten** stellen eine autonome Computerschaltung mit 3 kByte mit dem ROM-residenten Betriebssystem, Bootstrap-Loader, Ein/Ausgaberoutinen und Fehlersuchprogrammen, 1 kByte RAM und Serienschnittstellen für Bildschirmkonsole oder Teletype dar.
Bei Ausführung des Anwenderprogramms läuft dauernd ein zusätzlicher 256 x 32 bit großer Speicher mit, der sämtliche Aktivitäten auf Adreß-, Daten- und Steuerbus registriert.
Jedes fehlerhafte Verhalten Ihres Anwendersystems (z.B. Setzen eines bestimmten Bits bei einem bestimmten Port) kann man nun dem Entwicklungscomputer in Form einer Haltepunkt-Bedingung vorgeben. Er hält an dieser Programmstelle an und zeigt die letzten 255 Busaktivitäten an.

Man hat also die Möglichkeit, sich die Gegenwart (also was zum Zeitpunkt des Auftretens des Fehlers "los" ist) und die Vergangenheit (also was den Fehler bewirkt hat) anzusehen und unmittelbar zu korrigieren, ohne dabei die ganze Prozedur des Korrigierens per Editor, Assemblierens und Testens durchführen zu müssen, geschweige denn sogar EPROM's löschen und neu programmieren zu müssen.
Danach kann die Programmausführung durch ein Konsolenkommando fortgesetzt werden.

Darüberhinaus gibt es durch einen Triggerausgang am Entwicklungssystem sogar die Möglichkeit, sich Analog- oder Digitalsignale zum Zeitpunkt einer bestimmten logischen Bedingung mit einem Speicheroszillographen anzusehen. Der ganze beschriebene Vorgang geschieht im übrigen unter Echtzeitbedingungen, so daß man nach postiviem Ergebnis der Test auch wirklich sicher ist, daß das System funktioniert.

Die **Echtzeittesteinheit**
(bestehend aus 1 Echtzeitspeichersteckkarte und 1 Haltepunktsteckkarte) für Echtzeittests bietet rationellste Fehlersuch- und Beseitigungshilfsmittel für Hard- und Software durch Setzen von Hardwarehaltepunkten ("Breakpoints") in Abhängigkeit von Adress-Daten- oder Steuerwortkonstellationen, die das Entwicklungssystem beim Auftreten der vorgegebenen Bedingung in den Monitor-Mode bringen, in dem die ROM-residenten Fehlersuchprogramme eine Anzeige sämtlicher CPU-Register oder Speicherstellen und außerdem deren Änderung vor Fortsetzen der Anwenderprogrammausführung ermöglichen.

Die Echtzeit-Speicher-Steckkarte der Echtzeittesteinheit besteht im wesentlichen aus einem 256 x 32-bit Speicher, in dem bis zu 255 Computerzustände gespeichert werden. Jeder Zustand beinhaltet die momentane 16-bit-Adresse des Anwendersystems, seine 8-bit-Datenbusinformation und 7 Steuerbits (Control-Bus). Das verbleibende der 32 abgespeicherten Bits ist zur Identifikation des Übergangs zwischen Users und Monitor-Mode bestimmt.

Folgende Arten von Anwendersystem-Aktivitäten bzw. eine Kombination von ihnen lassen sich selektiv aspeichern und anzeigen:

- Lesezugriffe auf Speicher,
- Schreibzugriffe auf Speicher
- Lesezugriffe auf I/O-Ports.
- Schreibzugriffe auf I/O-Ports.

Die gleichen Aktivitäten sind, wie schon erwähnt, als Haltepuntkbedingungen zu verwernden, wobei als zusätzliche Bedingung Daten- und Adreßbusinformationen zu verwenden sind.

Dadurch lassen sich komplexe Ereignisse identifizieren, wie z.B. das Schreiben einer Eins auf Bitposition 6 des I/O-Ports Nummer 0FCH.

Speichersteckkarten

Der Arbeitsspeicher ist in 16 kByte-Modulen mit dynamischen Schreib/Lesespeichern aufgebaut, der maximal mögliche Ausbau beträgt 60 kByte.

Standardmäßig sind die minimal erforderlichen 16 kByte mitgeliefert.

Entsprechend der gewählten Betriebsart (Users- oder Monitor-Mode) enthält das Entwicklungssystem-RAM ein Anwenderprogramm, wobei dieses Zugriff auf die Entwicklungssystemperipherie hat (User-Mode) bzw. die zur Entwicklung des Anwenderprogramms nötige Software wie Editor, Assembler usw. (Monitor-Mode). Das Betriebssystem greift im Users Mode übrigens nicht auf die "unteren" 60 kByte des Entwicklungssystem-Speichers ein, so daß der Anwender frei über diesen Speicher verfügen kann.

Selbstverständlich kann dieser Bereich in jeder beliebigen Kombination externer RAM, ROM und PROM-Speicher belegt werden.

Bei Verwendung des Echtzeitadapters können diese externen Speicher auch noch mit dem systemeigenen RAM gemischt werden.

Die Floppy-Disk-Steuerkarte

Diese bedient 2 Floppy-Disk-Laufwerke, die als peripherer Speicher (hauptsächlich für die Betriebsprogramme und Zwischenspeicherung von Anwenderdateien) für das Entwicklungssystem dienen.

Die zum Urladen ("Bootstrap-Loader") und Ansteuern der Floppy-Disks nötige Software ist im ROM-residenten Betriebsprogramm ("Monitor") abgespeichert.

Über den Testadapter sind Adreßbus (16 bits), Datenbus (8 bit), sämtliche CPU-Steuersignale, der Systemtaktimpuls und Speicherfreigabesignal verfügbar. Sämtliche Leitungen des Adapters sind gepuffert und voll TTL-kompatibel.

Das Floppy-Disk-basierende Betriebssystem ist in der 1. Seite (= 4 kByte) in Festwertspeicher nichtflüchtig abgelegt.

6.2. Zusätzliche Steckkarten für das Entwicklungssystem

Die universelle Parallel/Ein/Ausgabe-Steckkarte ermöglicht durch ihre bi-direktionalen Ein/Ausgabebausteine Z80-PIO den Anschluß praktisch sämtlicher Arten peripherer Geräte mit Parallelschnittstelle an das Zilog-Entwicklungssystem; die Abwicklung der Ein/Ausgabe erfolgt hierbei durch entsprechende gerätespezifische Bedienprogramme, wie z.B. für PROM-Programmiergeräte, Drucker und Lochstreifengeräte.

Experimentierplatinen sind unbestückte Platinen in gleichem mechanischem Format wie die Entwicklungssystemsteckkarten, auf die zusätzliche, anwenderspezifische Schaltungen in Wire-Wrap- oder Fädeltechnik aufgebaut und in das Entwicklungssystem (beispielsweise zu Testzwecken) eingeschoben werden können; dabei kann u.a. die Spannungsversorgung des Zentralgerätes zur Speisung der Anwenderschaltung mitbenutzt werden.

Eine **Verlängerungsplatine** verwendet man, wenn eine der ins Zentralgerät eingesteckten Karten elektromechanisch von außen zugänglich gemacht werden soll. Dabei wird der Gerätedeckel abgeschraubt und die Verlängerungsplatine zwischen Steckfassung und der Platine, die aus dem Gerät herausragen soll, eingesteckt.

6.3. Die Floppy-Disk-Einheit

besteht aus

- einem Floppy-Disk-Doppellaufwerk, in dem die linke Diskette für Anwender, die rechte für Systemprogramme bestimmt ist und für diese beiden Laufwerke je einer Floppy-Disk-Ansteuerungskarte, die die Versorgung der Schrittmotoren, Antriebsmotorsteuerung und Sektormarken-Erkennung besorgt.

6.4. Systemkonsole

Als Systemkonsole ist ein Datensichtgerät zu empfehlen; in ihm verbinden sich die Vorteile eines geräuscharmen Dialogs mit minimalem Wartungsaufwand bei niedrigem Preis. Das Gerät ist direkt ohne Hard- oder Software-Zusätze an das Zentralgerät anschließbar. Stattdessen läßt sich natürlich auch die bekannte Teletype oder ein Thermodrucker (z.B. COMPUTER DEVICES "Minitherm") verwenden. In diesen Fällen hat man den Vorteil eines Papierausdrucks durch die Konsole, muß aber deren niedrige Geschwindigkeit in Kauf nehmen.

6.5. Der Echtzeit-Testadapter

besteht aus folgenden Elementen:

- einem 40 pin-Stecker, der während des Tests **statt** der Anwender-CPU in die Anwenderschaltung eingesteckt wird und
- einer Testadapter-Steuerungssteckkarte, die im Zentralgerät untergebracht ist. Dieser Zusatz, der übrigens mit dem Z80-Entwicklungssystem standardmäßig mitgeliefert wird, erlaubt im Austesten des gesamten Anwendersystems (also Hard- und Software) unter Echtzeitbedingungen. Hierbei wird das Programm im "User Mode" von der CPU im Entwicklungssystem verarbeitet, die wahlweise auf Speicher und Peripheriedes Entwicklungssystems oder der Anwenderschaltung zugreifen kann.

6.6. Schnelldrucker

Zur Ausgabe von Programm-Listing (Dokumentation!) ist ein Drucker erforderlich. Dessen Druckgeschwindigkeit hat bei Anwendern, die längere Programme erstellen, einen nicht zu vernachlässigenden Einfluß auf die Entwicklungsdauer, da der Entwickler erfahrungsgemäß während des Druckvorgangs vor seiner Anlage sitzen bleibt und in dieser Zeit kaum zu anderen Aktivitäten kommt.

Daher ist der HOUSTON-Schnelldrucker 8210/M sehr beliebt. Zu seiner Ansteuerung sind zwei der 4 Parallel-I/O-Ports der Ein/Ausgabe-Platine nötig. **Er erreicht eine Druckgeschwindigkeit von 2400 Zeilen/Minute.** Eine langsamere aber billigere Alternative stellt der neue Drucker von OKIDATA dar. Auch für ihn ist Interfaceschaltung und -Programm zum Anschluß an das Z80-Entwicklungssystem verfügbar (110 Zeichen/sec.).

6.7 PROM-Programmgerät

Wie erwähnt, werden direkt ins Entwicklungssystem einsteckbare Universal-Programmbaugruppen angeboten. Zur Off-Line-Programmierung von PROM's (was insbesondere dann sinnvoll ist, wenn große Mengen PROM's programmiert oder kopiert werden sollen und während der Progammierzeit das Entwicklungssystem nicht belegt werden soll) eignen sich Programmiergeräte, für die ein breites Spektrum an Personality-Moduln für praktisch sämtliche handelsüblichen PROMs verfügbar ist.
Diese Geräte haben einen eigenen Speicher, in den man die zu programmierenden Daten über eine Kabelverbindung, die nach der Datenübergabe getrennt werden kann, oder aber über einen Datenträger wie Magnetbandkasette oder Lochstreifen eingelesen werden können.
Der Speicherinhalt kann nun (z.B. von einer Hilfskraft) in beliebig viele PROM's einprogrammiert werden. Mit dem Z80-Entwicklungssystem ohne Entwicklungsaufwand verwendbare Universalprogrammiergeräte gibt es von den Herstellern DATA I/O, KONTRON und PROLOG.

7. Software für Entwicklungssysteme

Die folgende Beschreibung ist nur als grober Überblick gedacht. (siehe Bild 78) Einzelheiten finden sich im Z80-Entwicklungssytem-Software-Manual.

Das Floppy-Disk-basierende Betriebssystem OS Z80 ist in 1 Seite (= 4 kByte) Festwertspeicher nichtflüchtig abgelegt.

Über OS Z80 wird die Bedienung des Entwicklungssystems und das Umschalten zwischen Monitor- und User-Mode durch eine Reihe von Kommandos abgewickelt; Editor , Dateiverwaltungs- und Übersetzungsprogramme sind nicht in seinem Festwertspeicherbereich enthalten, sondern werden bei Bedarf von Diskette abgerufen.
Nach dem Einschalten des Gerätes ("Kaltstart") ermittelt der Computer beim Empfang eines ersten Zeichens von der angeschlossenen Konsole automatisch die Übertragungsrate der angeschlossenen Systemkonsole, richtet sich auf sie ein und quittiert das Kommando mit der Antwort "OS" auf der Systemkonsole. Das Entwicklungssystem befindet sich nun im "OS-Level". Es ist zum Arbeiten in einem der folgenden 4 Operations-Levels bereit:

- Editor level
- File maintenance level
- Assembler level
- Debug level

Die Kommandos dieser 4 levels sind im folgenden kurz beschrieben. Alle diese Kommandos können formlos ("free-form-format") eingegeben werden, d.h. Leerzeichen oder Kommas zwischen Kommando- und Datenfeld werden ignoriert und können daher in beliebiger Anzahl eingefügt werden.

Die Kommandos können voll ausgeschrieben oder aber abgekürzt werden. Zwei Sonderzeichen sind für Eingabekorrektur reserviert und können jederzeit verwendet werden:

@ Letztes Zeichen löschen
! Ganze Zeile löschen.

7.1. Editor

Der Z80-Texteditor wird durch das Kommando

EDIT dateiname dateityp

aufgerufen. Das Aufrufen des Texteditors veranlaßt gleichzeitig das Lesen der gewünschten Datei von der Diskette in den Entwicklungssystem-Arbeitsspeicher. Der Texteditor arbeitet mit Zeigern, über die der Entwickler auf die gewünschte Zeile zugreifen und sie mit einem der folgenden Kommandos modifizieren kann:

Hier ist ein Auszug aus seinem Kommando-Vorrat:

AGAIN	wiederholt das letzte Kommando
BOTTOM	bringt den Zeilenzeiger auf die unterste Zeile,
CHANGE	spricht eine gewünschte Zeichenfolge an und ersetzt sie durch eine von der Systemkonsole neu eingegebene Zeichenfolge. Es kann festgestellt werden, wie oft pro Zeile und wieviel Zeilen diese Änderung im Höchstfall erfolgen soll.
DELETE	löscht die angegebene Anzahl Zeilen aus der Datei.
FIND	sucht eine Zeile mit vorgegebenem Zeilenanfang auf

FILE	schreibt die Datei aus dem Arbeitsspeicher auf die Diskette und bewirkt die Rückkehr in den OS-Level.
GET	lädt einen auf der Diskette zwischen gespeicherten Text-Block in den Arbeitsspeicher.
GOTO	bringt den Zeiger auf die angegebene Zeilennummer.
INSERT	bewirkt das Einfügen eines von der Systemkonsole eingegebenen Textes zwischen der momentan angesprochenen Zeile und der darauf folgenden.
LINENO	Ausgabe der angegebenen Zeilennummer.
LOCATE	sucht eine bestimmte Zeichenfolge im Text und setzt den Zeilenzeiger auf die Nummer der ersten Zeile, in der diese Zeichenfolge vorkommt.
MACRO	verkettet eine Folge von mit "&"-Zeichen aneinandergereihter Editor-Kommandos unter dem angegebenen Namen, unter dem die gesamte angegebene Kommandofolge später aktiviert werden soll. Die Aktivierung erfolgt dann über das XECUTE-KOMMANDO (s.
NEXT	erhöht den Zeilenzeiger um den angegebenen Wert.
PRINT	bewirkt Ausgabe der nächsten n Zeilen auf die Kommandokonsole
PUT	lagert den angegebenen Textblock aus dem Arbeitsspeicher auf Diskette aus. Auf den so zwischengespeicherten Block kann dann über "GET" wieder zugegriffen werden.
REPLACE	ersetzt die laufende Zeile durch den folgenden eingegebenen Text.
SAVE	"rettet" den Inhalt des Arbeitsspeichers auf Diskette und bewirkt Rücksprung ins Editorprogramm.
TOP	bringt den Wert des Zeilenzeigers auf die erste Zeile.
UP	erniedrigt den Wert des Zeilenzeigers um die angegebene Zahl.
XECUTE	bewirkt Ausführung einer spezifizienten über MACRO aufgebauten Kommandofolge.

7.2. File maintenance level

In diesem Operationslevel erfolgen Datenmanipulationen. Es stehen u.a. folgende Kommandos zur Verfügung.

APPEND	Verkettung zweier Dateien
COMBINE	Faßt beliebig viele bestehende Dateien zu einer neuen Datei zusammen.
COPY	Kopiert den gesamten Inhalt einer Diskette auf eine andere.
ERASE	löscht eine Disketten-Datei.
FORMAT	initialisiert eine Diskette
LIST	Gibt Dateiübersicht aus.

NAME	ändert den Namen der angesprochenen Datei in den angegebenen anderen Namen.
PRINT	Ausgabe eines Dateiinhaltes auf die Systemkonsole im ASCII-Code.
QUIT	Rücksprung in den OS-Mode
STAT	Angabe der auf der Diskette unbenutzten Sektoren.

7.3. Assembler Level

Die Assembler des Z80-Entwicklungssystems sind echte Makroassembler; es ist ein "Absoluter" und ein "Relotativer" Assembler verfügbar. Der Aufruf des Assembler-Programms hat das Format

ASSEMBLE dateiname dateityp

Das Primärprogramm ("Quellprogramm") muß entsprechend den Regeln der Z80-Assemblersprache geschrieben sein, wobei die einzelnen Befehle folgendes Format haben:

marke: befehlscode operand-1 operand-2; kommentar

Zwischen den einzelnen Feldern sind beliebig viele Leerzeichen und Kommata zugelassen.
Marken werden mit einem Doppelpunkt abgeschlossen.
In Spalte 1 darf kein Befehlscode stehen.
Kommentare beginnen mit einem Strichpunkt und dürfen in jeder Zeile vorkommen.

7.4. Debug-Level

In diesem Level können Anwender-Programmabläufe analysiert, beeinflußt und korrigiert werden, wobei die Anwenderprogramme im Speicherbereich von 0 - 60 kByte des Entwicklungssystems stehen dürfen.
Die Ausführung eines Anwenderprogramms wird in diesem Level einfach durch das Kommando "GO" veranlaßt.

Im Debug-Level stehen u.a. folgende Kommandos zur Verfügung:

BASE	legt die Basis fest, in der Adressen und Daten gehandhabt werden (BCD, Hexadezimal).
BREAK	setzt einen automatischen Hardware-Haltepunkt (="Breakpoint") in der Echtzeit-Testeinheit. Als Haltepunkt-Bedingungen sind Speicher- bzw. I/O-Port-Zugriffe, (Lesen bzw. Schreiben) unter Angabe von Adreß-, Daten- und Datenmaskenbedingungen anzugeben.

Ein Anhalten der Ausführung des Anwenderprogramms ist daneben durch Betätigen der Monitortaste an der Frontplatte des Entwicklungssystems möglich. In beiden Fällen werden die CPU-Daten zu Analysezwecken abgespeichert.

DISPLAY gibt den Inhalt eines ausgewählten oder sämtlicher CPU-Register oder eines zusammenhängenden Speicherbereichs aus.

GO veranlaßt Ausführung eines mit LOAD geladenen Anwenderprogramms, und zwar ab jeder gewünschten Adresse oder ab einem Haltepunkt.
Zur Durchführung weiterer Kommandos muß nach Ausführung des GO-Kommandos ein Haltepunkt wirksam geworden sein.

HISTORY gibt auf der Bedienkonsole die letzten 255 Adreß-, Daten- und Steuerwerte Wirksamwerden des letzten Haltepunkts aus.

LOAD lädt ein Objektprogramm von der Diskette in den Entwicklungssystem-Arbeitsspeicher. Dieses Anwenderprogramm kann nun mit GO gestartet werden.

MOVE transferiert den Inhalt eines beliebig großen Speicherbereiches in einen anderen Speicherbereich.

PULSE wie BREAK jedoch wird das Programm bei Auftreten der Haltepunkt-Bedienung weiter ausgeführt und gleichzeitig extern ein Impuls (z.B. zur Synchronisation eines Oszillographen) ausgegeben.

QUIT Rücksprung in den OS-Level

SAVE speichert die im Arbeitsspeicher zusammengebundenen Programme und Unterprogramme auf Diskette ab.

SET bringt von der Systemkonsole eingegebene Daten in die angegebenen Register oder Speicherstellen.

STEP bewirkt Durchführung eines einzigen Befehls und anschließende Ausgabe der CPU-Registerinhalte.

TRACE legt fest, ob Speicher und I/O-Port-Zugriffe während der Programmausführung in der Echtzeit-Testeinheit festgehalten werden sollen.

An dieser Stelle soll hier nochmals eine bereits erwähnte besondere Fähigkeit des Z80-Entwicklungssystems kurz behandelt werden; es handelt sich dabei um eine Testhilfe, die durch den Echtzeittestmodul, den Echtzeittest-Speicher und die DEBUG-Kommandos "BREAK", "TRACE" und "HISTORY" implementiert ist. Vor dem Ablaufenlassen des Anwenderprogramms im "User-Mode" auf dem Entwicklungssystem kann man zunächst mit dem TRACE-Kommando festlegen, welche Busaktivitäten (man hat die Wahl zwischen einer beliebigen Kombination der Busaktivitäten "Speicher Lesen", "Speicherschreiben", "Ein/Ausgabe-Lesen und "Ein/Ausgabe-Schreiben") beim folgenden Test verfolgt werden sollen.

Außerdem formuliert man mit Hilfe des BREAK-Kommandos eine komplexe Testbedingung, bei deren Auftreten man die Ausführung des Anwenderprogramms abbrechen will (z.B., wenn Bit 3 des Ein/Ausgabekanals 127 gesetzt wird).
Das über "GO" zur Ausführung gebrachte Programm läuft nun solange ab, bis diese Bedingung auftritt; beim ganzen Ablauf werden sämtliche im TRACE-Kommando spezifizierten Aktivitäten auf Adreß-, Daten- und Steuerbus sukzessiv unter Echtzeitbedingung im Echtzeittest-Speicher (also nicht im Arbeits-Speicher!) nach dem Schieberegister-Prinzip abgelegt, so daß immer die letzten 255 Busaktivitäten in diesem Speicher stehen bleiben.
Nach Erreichen des Breakpoints kann man sich mit dem Kommando TRACE diese Aktivitäten ansehen und somit feststellen, wie die Breakpointbedingung (=Fehlverhalten des Systems) zustande kam.

Man kann jetzt die Fehlerursache durch direktes Setzen von Speicherstellen oder CPU-Registern beseitigen und das Programm mit GO oder auch im Einzelschrittverfahren weiter ausführen lassen und sämtliche dabei auftretende Fehler beseitigen, ohne in den Editor zurückzugehen, dort korrigieren und neu assemblieren zu müssen! Die eigentliche Programmkorrektur erfolgt erst am Schluß.
Dies und die Tatsache, daß man PROM's nur ein einziges Mal (nämlich nach vollständig durchzogenem Test) programmieren muß, bedeutet eine ungeheure Zeitersparnis.

7.4. Höhere Sprachen

Es sind ein residenter PL/Z-Compiler und in Kürze auch residente Compiler für COBOL, FORTRAN und PASCAL verfügbar. Darüberhinaus gibt es einen BASIC-Interpreter. Beschrieben werden diese Programme in einer getrennten Veröffentlichung.

Interpreter sind Programme, die Befehle in einer höheren Sprache akzeptieren, jedoch im Gegensatz zu Compilern keinen Objekt-Code generieren. Daraus folgt, daß bei Verwendung eines BASIC-Interpreters zum Ablauf eines BASIC-Anwenderprogramms grundsätzlich beide Programme (Anwenderprogramm **und** Interpreter) im Arbeitsspeicher resident sein müssen.

8. Beurteilung und Bewertung von MC

Bevor überhaupt die Entscheidung getroffen wird, daß ein Gerät auf Mikrocomputerbasis realisiert werden soll, muß geprüft werden, ob überhaupt der Einsatz irgendeines Mikrocomputers angebracht ist.
Es existieren nämlich immer noch folgende 4 Lösungsalternativen für Digitalsysteme:

- Hardwired Logik mit Standardschaltkreisen
- Kundenspezifischer Schaltkreis
- Einsatz eines Mikrocomputers
- Verwendung eines Minicomputers

Anzeichen, daß Wirtschaftlichkeitsoptimum erreichbar ist, liegen dabei vor für:

- Hardwired Logic mit Standardschaltkreisen bei niedrigen Stückzahlen und/oder
- Platzbedarf und Leistungsverbrauch unwesentlich und/oder
- Geringe Komplexität des Verarbeitungsproblems (also logische, aber wenig arithmetische Operationen erforderlich) und
- Problemstellung gleichbleibend, also Ein-/Ausgangsinformationen immer dieselben und
- keine Konzeptänderung zu erwarten
- Kundenspezifische Integrierte Schaltung bei sehr hohen Stückzahlen und/oder
- Platzbedarf und Leistungsverbrauch müssen minimal sein und
- Problemstellung gleichbleibend und
- keine Konzeptänderung zu erwarten und/oder
- maximale Verarbeitungsgeschwindigkeit nötig

o Mikrocomputer

- hohe Komplexität des Problems oder
- Notwendigkeit arithemtischer Operationen (z.B. Additionen, Zahlen/Maßstabsumwandlungen) oder
- wechselnde Ein-/Ausgabedaten
- Änderung der Verabeitung zu erwarten oder
- universelle Einsetzbarkeit der Hardware gefordert (= gleiches Grundgerät für verschieden Probleme) oder
- hohe Anforderungen an die Zwischenspeicherfähigkeit des Systems

o Minicomputer

- Anforderungen wie bei Mikrocomputer, jedoch Leistungsfähigkeit verfügbarer Mikrocomputer-Systeme nicht ausreichend. Zu bemerken ist dazu jedoch, daß

die Leistungsfähigkeit moderner "Mikros" (wie Z80A) durchaus an die von Minis herankommt, d.h. der Unterschied wird an dieser Stelle laufend geringer.

In Bild 79 ist schematisch der Weg zur Entscheidungsfindung dargestellt, ob ein konkretes Problem besser mit Standard-Logikbausteinen, einer kundenspezifischen integrierten Schaltung, einem Mikrocomputersystem oder durch Verwendung eines Minicomputers zu lösen ist.

Der Vorgang der Entscheidungsfindung ist dabei so klar dargestellt, daß hier nur noch die Gründe zur Wahl der einen oder anderen Alternative erläutert werden sollen:

1: Wenn nur wenige Gerätefunktionen zu realisieren sind, ist das Aufbieten von Computerfähigkeiten meist überflüssig.

2: Zur Verifizierung komplexerer Funktionen (zum Beispiel mathematischer Auswertungen eines Meßergebnisses) eignen sich festverdrahtete Schaltungen grundsätzlich verhältnismäßig schlecht (großer Bauteilaufwand).

3: Die sequentielle Verarbeitungsweise von Mikroprozessoren begrenzt ihre Einsetzbarkeit auf Anwendungen nicht zu hoher Reaktionsgeschwindigkeiten ($< \mu s$).

4: Wo spätere Änderungen vorhersehbar sind, ist eine festverdrahtete Lösung grundsätzlich nicht vorteilhaft (schwer zu ändern!).

5: Programmierbarkeit größeren Stils ist nur in Computeranordnungen möglich.

6: Zwischenspeicherung kann besonders gut von Computerschaltungen durchgeführt werden.

7: Die Entwicklung kundenspezifischer Integrierter Schaltungen lohnt sich wegen der hohen Entwicklungs-(-fix)kosten nur bei Abnahme sehr hoher Stückzahlen.

Ist an dieser Stelle die Entscheidung zugunsten des Mikrocomputers gefallen (was durch den technologischen Fortschritt und die sinkenden Mikrocomputer-Bausteinpreise in einer wachsenden Anzahl von Anwendungen der Fall ist) folgt meist die Frage:

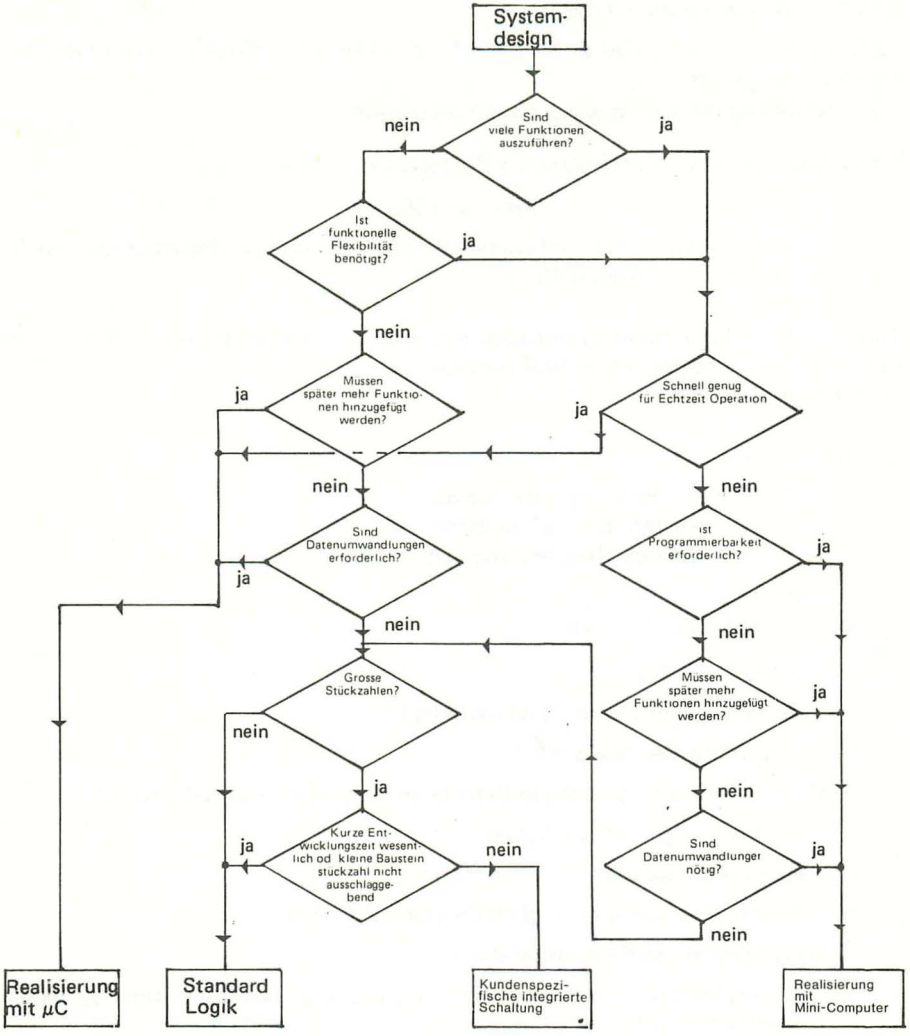


Bild 79: Weg zur Entscheidungsfindung bei der Mikrocomputer Auswahl

Welcher Mikrocomputer ist der Beste?

Dazu ist zu bemerken, daß es für diese Frage keine vernünftige Antwort gibt, da sie falsch gestellt ist.
Die Anwendung der häufig exerzierten Kalkulation

$$\text{Vergleichskriterium} = \frac{\text{Taktfrequenz x Prozessorwortlänge}}{\text{Preis der CPU}}$$

ist lediglich dem technischen und wirtschaftlichen Unverstand **dessen** proportional, der nach ihr Entscheidungen trifft.

Tatsächlich sind die heranzuziehenden Kriterien problemabhängig und daher von Fall zu Fall unterschiedlich zu beantworten.
Solche Kriterien sind:

- ☐ Befehlsvorrat
- ☐ Befehlssatz + Adressierung wirkungsvoller:
  - → kürzere Entwicklungszeit
  - → kürzere Verarbeitungszeit
- ☐ Architektur
  Wichtig in der CPU-Struktur:
  - o Anzahl der Akkus
  - o Arbeitsregister (scratch pad momory)
  - o Zugriff auf "Adress Stack"

  Einfluß auf Anzahl der nötigen Befehle im Programm und dadurch auf
  - o Verarbeitungsgeschwindigkeit
  - o Entwicklungskosten
  - o Bauelementekosten bezüglich Programmspeicher
- ☐ Geschwindigkeit wird bestimmt durch
  - o durchschnittliche Befehls-Ausführungsdauer ("execution time"), diese wiederum durch
  - o "Execution Time" der am häufigsten verwendeten Befehle (anwendungsspezifisch!) diese wiederum durch
  - o Anzahl der Maschinenzyklen pro Befehl.
  - o die Dauer eines Maschinenzyklus (~ Taktfrequenz).

Die Geschwindigkeit von Mikrocomputersystemen vergleicht man am besten anhand der Ausführungszeit gleichartiger Standardprogramm, sogenannter "BENCHMARKS". Problematisch daran ist allerdings, daß Benchmarks

- ☐ durch die Auswahl der Problemstellung immer noch subjektiven Charakter haben und daher vom Anbieter beeinflußt werden können und

- ☐ durch die Auswahl der Problemstellung anwendungsspezifisch sind und evtl. am tatsächlich vorliegenden Problem vorbeigehen.

Durch hohe Geschwindigkeit kann das Gerät mehr Aufgaben in der gleichen Zeit lösen, als ein langsameres.

- ☐ Interrupt-Möglichkeiten
  Besonders wichtig für Probleme, bei denen unregelmäßig Informationen anfallen, der Rechner aber im "Background" eigentlich dauernd beschäftigt wäre!
- ☐ Schnittstelleneigenschaften
  - o Getrennter Daten- und Adreß-Bus?
  - o Quittungs-Betrieb (Handshaking) möglich?
  - o Wie sehen die Ausgänge aus?
    (TTL-Kompatibel? Ausgangsbelastbarkeit (fan out)?)

Direkter Einfluß auf die Systemkosten durch weniger Bauelementeaufwand!

- ☐ Speicherkonzept
  - o Anschlußmöglichkeit von Standard-Speichern?
  - o Anschlußmöglichkeit von langsamen Speichern für "langsame" Anwendungen (kostengünstige Speicher-Realisierung)
    Einfluß auf Systemkosten!
- ☐ Versorgungsspannungen
  Je weniger, desto besser!
  Effekt auf Systemkosten. Platzbedarf.

  Optimum: Eine einzige Spannungsquelle
  Allerdings: Es nützt nicht viel, wenn nur die Minimalkonfiguration mit einer Versorgungsspannung auskommt!
- ☐ Verfügbarkeit von Bauelementen
  - o Ist das Systemspektrum abgerundet?
  - o Sind die Bausteine tatsächlich käuflich oder nur angekündigt?
- ☐ Sicherung der Lieferfähigkeit (Zweithersteller)
- ☐ Zusatz Hard/Software für Systementwicklung
  - o Mikrocomputer-Entwicklungssystem
  - o Testgeräte
  - o Peripherie (Datensichtgerät, Drucker, Floppy-Disk)
  - o PROM Programmiergerät
  - o Prototyp Platinen bzw. Kits (=Bausätze)
  - o Entwicklungssoftware für Programmaufbau (Editor), Übersetzung (Assembler, Compiler), Fehlersuche (Simulator)
  - o Standardprogrammbibliothek

- ☐ Dokumentations- und Ausbildungsangebot
  - o Ausbildungsmöglichkeit
  - o Bauelemente-, System- und Softwarebeschreibungen verfügbar?
  - o Applikationsbereiche
- ☐ Applikatiorische Unterstützung des Anwenders

  System-Preis

  (beinhaltet nicht nur den Preis der CPU, sondern vor allem auch die Speicherkosten! Hilfs-Logikschaltungen mit berücksichtigen)

  Zukunfssicherheit des Herstellerkonzepts
  (softwarekompatibles Spektrum von Prozessoren unterschiedlicher Leistungs- und Preisklasse, Weiterentwicklung auf dem Software- und dem Systemsektor).

Grundsätzlich zum ganzen Auswahlvorgang zu bemerken ist jedoch folgendes:

Man kann keinesfalls für jedes anstehende Realisierungsproblem einen anderen Prozessor wählen, da

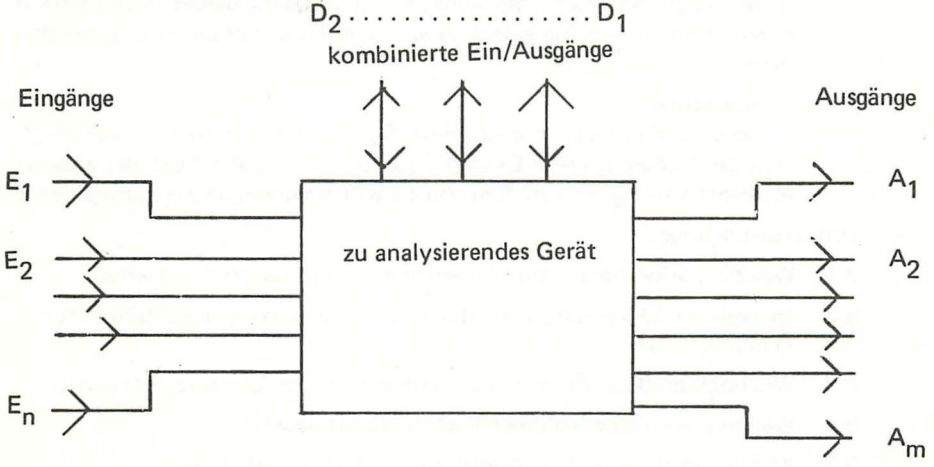
- der Aufwand zur Einarbeitung der Entwickler in eine Prozessorfamilie dazu im allgemeinen zu hoch sein wird und
- die Investitionen auf der Seite der Entwicklungshilfen (Entwicklungssystem) nicht jedesmal neu getroffen werden können.

Ausgewählt werden sollte also **die** Prozessorfamilie, die technisch **und** wirtschaftlich am zukunftssichersten erscheint.

ANHANG 1

Checkliste zur Mikrocomputer-Problemdefinition

1. Ist personelle Kapazität zur Realisierung des Geräts verfügbar?
   1.1. Entwicklung im Hause?
   1.2. Auftragsentwicklung durch Fremdfirma?
2. Sind hierfür (1.1.) Investitionen zu treffen?
3. Kann das Gerät gewartet werden?
4. Spätester Zeitpunkt der Gerätefertigstellung
5. Angaben zur Realisierung:
   5.1. Existiert bereits eine Problemlösung und in welcher Form (z.B. TTL)?
   5.2. Welche Stückzahl/Jahr kommt in Frage?
   5.3. Wie hoch ist der max. tragbare Bauelementepreis pro Gerät (ohne Stellglieder wie Relais, Triacs usw.)?
   5.4 Aus welchem Grund soll eine Realisierung mit Mikrocomputer erfolgen (z.B. Flexibilität, Senkung der Bauteilekosten)?
6. Ein/Ausgabe-Analyse:
   Bitte fertigen Sie eine Liste sämtlicher Schnittstellen an, über die Ihr Gerät mit der Umwelt in Kontakt sein soll.



   6.1. Eingänge veranlassen Reaktionen des Geräts oder geben ihm Informationen (Beispiele. Meßgröße, Alarmsignal, Betriebsartenschalter).
   6.2. Ausgänge liefern die Reaktionen des Geräts (Beispiele: Anzeigen, Leistungsschalter).

6.3. Ein/Ausgänge:
Schnittstellen, an denen sowohl Ein- als auch Ausgaben über die gleichen Leitungen erfolgen (Beispiele: Fernschreiberleitung, bidirektionaler Datenbus).

6.4. Erläuterungen:
Ihr System muß nicht zwangsläufigerweise alle diese Schnittstellenarten beinhalten. Für jede der aufgelisteten Ein-, Aus- bzw. Ein/Ausgänge sind folgende Angaben wichtig:

–Bezeichnung und Zweck

- Auf wievielen Leitungen laufen die Signale eines Ein-, Aus-, oder Ein/ Ausgang parallel (eine oder mehrere Leitungen)?
- In welchem minimalen zeitlichen Abstand müssen die Signale aufeinander folgen können?
- Falls die Signale zeitlich miteinander korreliert sind, sollte ein Zeitdiagramm beigefügt werden.
- Welche physikalischen Anforderungen werden an die Signale gestellt (Störsicherheit, angeschlossene Leitungsart/Länge, Potentialfreiheit, Strom/Spannung/Leitung, Normen wie TTL-Pegel oder V24-Schnittstelle).

7. Gerätestruktur

7.1. Blockschaltbild
Bitte geben Sie nach Möglichkeit ein Blockschaltbild entsprechend Punkt 6 an, in dem Sie jedoch in einzelne Funktionseinheiten untergliedern.

7.2. Verknüpfung
In welcher Form sind die einzelnen Signale miteinander verknüpft (z.B. A_{15} wird gesetzt, wenn E_1 und E_3 2 sec. ein Signal anliegt; der analoge Meßwert von E_4 wird zu dem von E_2 addiert und über A_3 ausgegeben)?

8. Sonderbedingungen

8.1. Welche Vorkehrungen sind für einen evtl. Stromausfall zu treffen?

8.2. In welcher Umgebung wird das Gerät eingesetzt (Temperaturbereich, Dämpfe, . . .)?

8.3. Wie hoch muß die Funktionssicherheit sein (ggf. Ersatzsystem usw.)?

8.4. Welches räumliche Volumen darf das Gerät haben?

8.5. Welche Spannungs-/Stromversorgung steht zur Verfügung?

ANHANG 2

Mikrocomputer-Entwicklungssystem

Technische Daten:

Zentraleinheit	Mikroprozessor Z80-CPU
Speicher:	3 kByte ROM für Betriebssystem 1 kByte stat. RAM reserviert 16 kByte dyn. RAM für allgemeine Verwendung bis 60 kByte erweiterbar.
Ein/Ausgabekanäle:	– Standard-Interfaces o Serienschnittstelle o Floppy-Disk-Schnittstelle o Echtzeit-Testadapter – Optionale Interfaces für o PROM-Programmiergerät o Zeilendrucker
Ausstattung:	Zentraleinheit-Steckkarte mit 4 kByte ROM/RAM und Betriebssoftware und RS 232 oder Strompräge-Serienschnittstelle – Eine 16 kByte dyn-RAM-Steckkarte – Echtzeit-Testeinheit (bestehend aus 1 Echtzeitspeichersteckkarte und 2 Haltepunktsteckkarte – Floppy-Disk-Steuerungssteckkarte für 1 Floppy-Disk Doppellaufwerk – Steckkarte für Echtzeit-Testadapter – Echtzeittestadapter zur Direktverbindung zwischen Zentralgerät und Anwenderschaltung für Echtzeit Hard- und Softwaretests ("Users Hardware Interface") mit Steuerungsplatine
Mitgelieferte Software:	– ROM-residentes Betriebssystem und Testprogramme – Floppy-Disk-Betriebssystem – Texteditor – Datenverwaltungsprogramm – Assembler – Dokumentation

Optionale Ausstattung:	– Relokativer Assembler und Compiler – Parallel-Ein/Ausgabe-Steckkarte zum Anschluß von PROM-Programmiergerät, Drucker usw. – weitere 16 kByte-RAM-Steckkarten
Stromversorgung:	115 oder 230 V, 50/60 Hz, 200 W
Zulässige Umgebungstemperatur:	0 ...50° C
Elektromechanischer Aufbau:	Abmessung: ca. 483 x 229 x 381 mm Gewicht: ca. 17 kg
Stromversorgung:	Zusätzlich zur nötigen Stromversorgung der Grundausstattung sind über das Zentralgerätechassis bis zu 4 A bei + 5 V ± 5% entnehmbar.

Verlagsprogramm

Übersicht lieferbarer Bücher

Bestell Nr.	ISBN	Verfasser	Titel	Preis DM
1	3-921682-01-0	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 1	19,80
2	3-921682-02-9	Hofacker	Transistor Berechnungs- und Bauanleitungshandbuch, Band 2	19,80
3	3-921682-03-7	Gebauer	Elektronik im Auto	9,80
4	3-921682-04-5	Lorenz	IC-Handbuch, TTL, CMOS, Linear	19,80
5	3-921682-05-3	Steinbach	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	3-921682-06-1	Steinbach	IC-Schaltungen, TTL, CMOS, Linear	9,80
7	3-921682-33-9	Hofacker	Elektronik Schaltungen	5, –
8	3-921682-08-8	Lorenz	IC-Bauanleitungs-Handbuch	19,80
9	3-921682-09-6	Lorenz	Feldeffekttransistoren	5, –
10	3-921682-34-7	Lorenz	Elektronik und Radio	5, –
11	3-921682-11-7	Lorenz	IC-NF Verstärker	9,80
12	3-921682-12-6	Bernstein	Beispiele Integrierter Schaltungen (BIS)	19,80
13	3-921682-13-4	Lorenz	HEH, Hobby Elektronik Handbuch	9,80
14	3-921682-14-2	Lorenz	IC-Vergleichsliste	9,80
15	3-921682-15-0	Lorenz	Optoelektronik Handbuch	19,80
16	3-921682-16-9	Bernstein	CMOS Teil 1 Einführung Entwurf, Schaltbeispiele	19,80 19,80
17	3-921682-17-7	Bernstein	CMOS Teil 2 Entwurf und Schaltbeispiele	19,80
18	3-921682-18-5	Bernstein	CMOS Teil 3 Entwurf und Schaltbeispiele	19,80
19	3-921682-19-3	Lorenz	IC-Experimentier Handbuch	19,80
20	3-921682-20-7	Lorenz	Operationsverstärker	19,80
21	3-921682-21-5	Lorenz	Digitaltechnik Grundkurs	19,80
22	3-921682-22-3	Bernstein	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	3-921682-23-1	Lorenz	Elektronik Grundkurs Kurzlehrgang Elektronik	9,80
24	3-921682-35-5	Hans Peter Blomeyer-Partenstein	Mikrocomputer-Anwender-Handbuch MAH	29,80
25	3-921682-25-8	C. Lorenz	Hobby Computer Handbuch HCH	29,80
26	3-921682-26-8	H. Bernstein	Mikroprozessor Teil 2	19,80
27	3-921682-27-4	C. Lorenz	Mikrocomputer Software Handbuch MSH	29,80
28	3-921682-28-2	C. Lorenz	Lexikon + Wörterbuch für Elektronik und Mikroprozessortechnik LEM	29,80
29	3-921682-29-0	H. Bernstein	Handbuch für theoretische und angewandte Elektronik Teil 1	49,80
30	3-921682-30-4	H. Bernstein	Handbuch für theoretische und angewandte Elektronik Teil 2	49,80
31	3-921682-31-2	C. Lorenz	Basic Digital Electronics Course (englisch)	19,80
31	3-921682-32-0	C. Lorenz	Circuits Digital Et Pratique (französisch)	19,80
41			Experimentierplatine für 14, 16, 24, 28, und 40 pol. DIL ICs.	79, –

Ing. W. Hofacker GmbH Verlag
8 München 75

Best.Nr.

1

TBB - Handbuch Band 1 , W. Hofacker

Transistor Berechnungs- und Bauanleitungshandbuch Band 1

Das Handbuch für jeden Elektroniker. Rechenbeispiele, Berechnungsgrundlagen, Bauanleitungen, Nomogramme, Tabellen und Vergleichslisten aus den wichtigsten Bereichen der Elektronik. Ein Buch zum Einarbeiten in die Elektronik. Ein Buch zum Nachschlagen. Grundlagen Digitaltechnik, Netzgeräte und Transformatorenberechnung, Berechnung von Multivibratoren, Schmitt Trigger u. v. a. Über 130 Seiten. **DM 19,80**

2

TBB - Handbuch Band 2 , W. Hofacker

Transistor Berechnungs- und Bauanleitungshandbuch Band 2

Dieses Buch ist die Fortsetzung des erfolgreichen Handbuches TBB-Handbuch Band 1. Ein Buch, das sich in der Hand des Praktikers bestens bewährt hat. Weitere neueste Schaltbeispiele und Berechnungsgrundlagen. Experimentier- und Versuchsbeschreibungen. Integrierte Spannungsregler, Wärmeableitung, Operationsverstärker Einführung, RC-Zeitglieder, Transistortester u. v. a. **DM 19,80**

3

Elektronik im Auto , H. Gebauer

Ein Buch für jeden technisch interessierten Autofahrer. Es zeigt Ihnen die vielen Möglichkeiten zur Verbesserung von Sicherheit, Leistung und Fahrkomfort in Ihrem Auto. Thyristorzündung, Drehzahlmesser, Beschleunigungsmesser, Geschwindigkeitswarner, Batterieladegerät u. v. a. Tips, genaue Beschreibungen, Bauanleitungen. **DM 9,80**

4

IC - Handbuch , C. Lorenz

Handbuch für digitale und lineare integrierte Schaltungen.

Sensationelle Neuheit. Ein Handbuch für digitale und lineare Schaltkreise. Daten- und Auswahllisten, Vergleichslisten, Gehäuseformen, Grundlagen, Einführungsbeschreibungen, viele Schaltbeispiele, Bauanleitungen, Printvorlagen, u. v. a.
Alles über TTL-Technik, C MOS, MOS-Schaltungen, Uhren ICs, lineare Schaltungen, ein Chip-Rechner, integrierte NF-Verstärker u. v. a. **DM 19,80**

5

IC - Datenbuch , D. Steinbach

Daten- und Auswahllisten der gebräuchlichsten integrierten Schaltkreise. Digital und analog. Gerade bei ICs ist es wichtig die Anschlußfolgen genau zu kennen. Auf über 55 Seiten finden Sie: Die wichtigsten TTL-Schaltkreise, C-Mos Serie, lineare Schaltungen wie Operationsverstärker, Komparatoren, NF-Verstärker, Spannungsregler, Triggerschaltungen, Impulsgeber u. v. a. Weiterhin finden Sie eine C Mos-Vergleichsliste sowie Kurzdaten und logisches Verhalten dieser C MOS Elemente. Das IC-Datenbuch wird auch Ihnen ein unentbehrlicher Begleiter bei allen Arbeiten mit integrierten Schaltungen sein. **DM 9,80**

6

IC - Schaltungen , D. Steinbach

Hier finden Sie eine gelungene Zusammenstellung der wichtigsten Anwendungsbeispiele aus dem Bereich der integrierten Schaltungen. TTL – C MOS – Linear. Alle Schaltungen sind übersichtlich und klar dargestellt und mit einer kurzen, jedoch sehr genauen Beschreibung versehen. Viele Schaltungen sind Grundschaltungen, die man beim Umgang mit integrierten Schaltungen immer wieder benötigt. Tastenentprellung, Zähler, Impulsgeber, Codierer, Dekodierer, Datenübertragung, Serien-Parallel-Wandler, Digitalvoltmeter u. v. a. **DM 9,80**

7

Elektronik Schaltungen,
4. völlig neu überarbeitete Auflage, W. Hofacker

Die ideale Schaltungssammlung zum Basteln und Experimentieren. Schaltungen mit Operationsverstärkern, Spannungsreglern, TTL, C-MOS Schaltkreisen. MOS Uhr mit Wecker, elektronischer Würfel, Musik Synthisizer, Timer 555 Anwendungen, Experimentieranleitungen und viele andere hochinteressante Schaltbeispiele tlw. mit Printvorlage. 64 Seiten Inhalt. DM 5,–

8

IC - Bauanleitungs -Handbuch -IC -KIT , C. Lorenz

Ein Bauanleitungsbuch mit vielen hochinteressanten Bauanleitungen aus dem Bereich der LSI Schaltungstechnik. Schaltbeispiele mit Printvorlagen zum Selbstherstellen der Leiterplatten mit genauesten Beschreibungen. Hochaktuell und brandneu: Funktionsgenerator XR 2206, MOS-Uhr mit Wecker, Schlummerautomatik und programmierbarem Wecktongenerator, Sensortastenwahl, IC-Netzteil, Funktionsgenerator 8038 neuartige Transistorzündung, 35 W NF-Verstärker, Experimentieranleitung und Grundkurs über Flip Flops, Experimente mit Digitalschaltungen u. v. a. Zu allen Schaltungen finden Sie Platinvorlagen oder Sie können die Experimentierschaltungen auf der Experimentierplatine WH-1 g durchführen. Über 125 Seiten. DM 19,80

41

Experimentierplatine mit Sockel, Stecker und Füßen Typ WH-1g für 40,28,24,16 und 14 polige DIL -Gehäuse DM 79,–

9

Feldeffekttransistoren, C. Lorenz

Der Feldeffekttransistor (FET) gehört heute zu den interessantesten Bauteilen überhaupt. Wie man damit experimentiert, wie man seine Funktion versteht und wie man damit brauchbare und hochinteressante Schaltungen aufbauen kann, zeigt Ihnen dieses Buch. Grundlagen, Kennlinienfelder, Tabellen, Berechnungsgrundlagen, Rechenbeispiele, Anschlußbilder und eine Vergleichsliste für Feldeffekttransistoren bilden den Kern dieser umfangreichen Darstellung. Alles in allem finden Sie hier eine praxisnahe und komplette Arbeitsunterlage, mit der Sie im Beruf und auch im Hobby erfolgreich arbeiten können. Über 45 Seiten. DM 5,–

10

Elektronik und Radio , C. Lorenz

Eine Einführung in die Radiotechnik, wie man sie nicht alle Tage findet. Eine Einführung für jeden interessierten Amateur. Aber auch der Profi findet hier Berechnungsgrundlagen, Rechenbeispiele, Tabellen und Nomogramme. Viele Schaltbeispiele mit Sendern im MW-, KW- und UKW-Bereich sowie den zugehörigen Empfängerschaltungen. IC-Radio, IC-Sender und vieles andere mehr. Über 40 Seiten. DM 5,–

11

IC -Niederfrequenzverstärker , C. Lorenz

Grundlagen der integrierten NF-Verstärker, Berechnung von kompletten IC-NF-Verstärkerstufen. Anwendungsbeispiele mit den interessantesten und gebräuchlichsten Standard IC-NF-Verstärkern wie TBA 800, TBA 830, usw. Printvorlagen, Auswahltabellen, Experimentieranleitungen und Anschlußbilder machen dieses Buch zu einem unentbehrlichen Begleiter für alle, die sich mit NF-Verstärkern beschäftigen wollen. Über 65 Seiten. DM 9,80

12

BIS BUCH, Beispiele integrierter Schaltungen, H. Bernstein

Auf über 130 Seiten Anwendungsbeispiele mit integrierten Schaltkreisen. Zeitgeber 555, Funktionsgenerator ICL 8038, Opto Elektronik, Operationsverstärker, Analogschalter, Digital-Analog-Wandler, Analoge Rechenbausteine, Schreib-Lese-Speicher (RAM), Festwertspeicher (ROM), Speicherschaltungen, Uhrenbausteine u. v. a. **DM 19,80**

13

HEH, Hobby Elektronik Handbuch , C. Lorenz

Das Schaltungsbuch für jeden Hobbyelektroniker. Schaltbeispiele und Bauanleitungen aus dem gesamten Hobbybereich. Lichtorgeln, Eiswarngerät fürs Auto, Alarmanlagen, Metallsuchgerät, PLL-Schaltungen, Logik-Tester, Funktionsgeneratoren u. v. a. Über 55 Seiten. **DM 9,80**

14

IC - Vergleichsliste , C. Lorenz

Vergleichsliste für digitale und lineare integrierte Schaltkreise.

Standard TTL, Low Power Schottky TTL, C MOS, Triacs Thyristoren, Optoelektronik, Operationsverstärker, Spannungskomparatoren, Spannungsregler, NF-Verstârker u. v. a. Funktionsvergleichsliste CMOS zu TTL. Vergleichstabelle für Transistoren und Dioden sowie Darlingtontransistoren. Eine Vergleichsliste, die man immer wieder braucht. **DM 9,80**

15

Opto -Handbuch, Handbuch für Optoelektronik , C. Lorenz

Das Handbuch für die gesamte Optoelektronik. Eine Einführung und ein ideales Nachschlagwerk. Grundlagen, Definitionen aller Kenngrößen, Opto-Lexikon, Berechnungsgrundlagen, Rechenbeispiele, Schaltbeispiele: Lichtsender, Lichtempfänger, Anzeigen, Infrarot Detektoren, Lichtmeßgerät, Optokoppler, Pegelschalter, Opto-Vergleichsliste. Anschlußbilder wichtiger 7-Segment-Anzeigen u. v. a. Über 106 Seiten. **DM 19,80**

16

C MOS Einführung, Entwurf, Schaltbeispiele, Teil 1 , H. Bernstein

Vom C MOS Gatterbaustein über Schieberegister und Zähler bis hin zum C MOS Schreib-Lesespeicher. Insgesamt werden neunzehn interessante und bekannte C MOS Schaltkreise beschrieben. Zu jedem Bauelement sind genaue Daten, Schaltbild und Anwendungsbeispiele angegeben. Im großen Applikationsteil finden Sie: C MOS-Kippstufen, Addierwerke und Rechenschaltungen, Digital Analog Wandler, Schieberegister für analoge Spannungen, Multiplexsysteme für analoge Signale u. v. a. Eine komplette Einführung und gut geeignet für das Selbststudium der C MOS Technik. Über 140 Seiten. **DM 19,80**

17

C MOS Entwurf und Schaltbeispiele, Teil 2 , H. Bernstein

Fortsetzung von Teil 1. Anwendungsbeispiele mit genauen Schaltungsbeschreibungen und Bauelementeunterlagen. Daten, Anschlußbelegungen weiterer wichtiger hochintegrierter C MOS Elemente. Ein komplettes Arbeits- und Experimentierbuch. C-MOS Uhrenschaltungen, Schieberegisterschaltungen, Parallel-Serien Umsetzung, statische und dynamische Speicherschaltungen, Zählschaltungen, Digital Analog Wandler, Analog Digital Wandler, Digital Voltmeter, I/O Registerschaltungen, Codier und Dekodierschaltungen. RAM und ROM Anwendungen. Über 140 Seiten. **DM 19,80**

18

C MOS Entwurf und Schaltbeispiele, Teil 3 , H. Bernstein

Fortsetzung von Teil 2. Eine sehr umfangreiche Applikationssammlung mit hochintegrierten C MOS Elementen. Rechnerschaltungen, Speicher- und Steuerschaltungen, Multiplex- und Datenbussysteme, Uhrenschaltungen, PLL-Schaltungen, Liquid Cristal Anzeigen und deren Treiberschaltungen, Optoelektronik in Verbindung mit C MOS. Grundlagen, Aufbau und Wirkungsweise der Prozeßrechentechnik, Arithmetische Logische Einheiten (ALU) und andere wichtige Funktionen aus der Prozeßrechentechnik. RAMs, ROMs und FIFO-Speicherschaltungen. Über 140 Seiten. Lieferbar Frühjahr 1977. **DM 19,80**

19

IC Experimentier Handbuch - IC -EX, C. Lorenz C. Lorenz

Eine sehr umfangreiche Schaltungssammlung und Bauanleitunssammlung mit neuesten integrierten Bausteinen. Neue, jedoch beim Fachhandel erhältliche Standard ICs. Rechnerschaltungen, Mikroprozessoren, I/O Schaltungen, druckende und anzeigende Rechner, Stoppuhren, Zählerschaltungen, Digitalvoltmeter, professioneller Synthisizer, Hilfsschaltungen für den Elektronik Experimenter, Analog Digital Wandler, Frequenzzähler u. v. a. hochinteressante Bauanleitungen. Viele Schaltungen können auf der IC KIT Experimentierplatine WH–1g aufgebaut werden. Über 120 Seiten. Lieferbar Frühjahr 1977.

DM 19,80

20

Operationsverstärker, Grundlagen und Schaltbeispiele , C. Lorenz

Dieses Buch umfaßt das gesamte Gebiet der linearen Schaltungstechnik und stellt ein in dieser Preislage bisher noch nie dagewesen Nachschlagwerk und Einführungshandbuch dar. Bestens geeignet für das Selbststudium. Nach einer pädagogisch geschickt gemachten Einführung folgen theoretische Arbeitsunterlagen und die zugehörigen Schaltbeispiele mit Daten und Gehäuseanschlüssen. Dieses wertvolle Buch dürfte seinen Platz auch bei Ihren Arbeitsunterlagen finden, und wird dann immer von Nutzen sein, wenn es um die Lösung von nicht routinemäßigen Aufgaben geht. Lieferbar Frühjahr 1977. **DM 19,80**

21

Digitaltechnik Grundkurs (TTL -C MOS - MOS und Software) , C. Lorenz

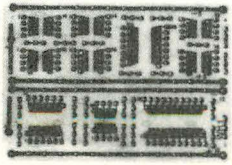
Ein Einführungskurs in die Digitaltechnik für Anfänger und Fortgeschrittene. Ein Fachbuch für den programmierten Selbstunterricht. Der ideale Kurzlehrgang für das Selbststudium. Der Kurs vermittelt Ihnen alle wichtigen Grundkenntnisse vom TTL-Gatter bis zum Mikroprozessor und Lösung von Schaltungsaufgaben durch Software. Viele Versuchsaufbauten und Experimente aus diesem Kurs können auf der IC-KIT Platine WH–1g durchgeführt werden. Grundlagen, Gatter, Zähler, programmierbare Zähler, IC-Tester, Schieberegister, Speicher, Mikroprozessoren u. v. a. Über 130 Seiten.
Lieferbar Frühjahr 1977.

DM 19,80

41

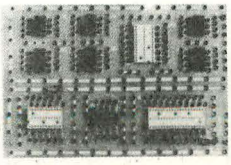
Experimentierplatine WH-1g dazu, Best.Nr. 41

DM 79,–



Experimentierplatine WH - 1 g
Abmessungen: 210 mm x 150 mm

Experimentierplatine WH - 1 g
fertig aufgebaut und mit Sockeln bestückt.



22

Mikroprozessoren, Eigenschaften und Aufbau Teil 1 , H. Bernstein

Grundlagen, Eigenschaften und Aufbau von Mikroprozessoren. Organisation von Recheneinheiten und Mikroprogrammen. Programmierung und Klassifizierung von Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein Chip-Technik und Multi Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs, ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA)
Anwendungsbeispiele und Anwendungsbereiche. Über 120 Seiten. **DM 19,80**

23

Elektronik Grundkurs (Kurzlehrgang Elektronik) , C. Lorenz

Eine leichtverständliche und pädagogisch geschickt gemachte Einführung in die Technik der elektronischen Schaltungen. Ein Kurzlehrgang und Schnellkurs zugleich. Aber auch ein recht brauchbares Nachschlagewerk für den fortgeschrittenen Elektroniker. Mit wenig Mühe können Sie sich hier die Grundkenntnisse der elektronischen Schaltungspraxis aneignen. Das Buch schafft die Voraussetzungen für ein erfolgreiches und sicheres Arbeiten mit interessanten Schaltkreisen modernster Technologien. Unentbehrlich für das Experimentieren mit den heutigen modernen hochintegrierten Schaltkreisen. Über 150 Seiten.
DM 9,80

24

Mikrocomputer Anwender Handbuch MAH, Hans Peter Blomeyer-Partenstein

Entwicklung und Einsatz von Mikrocomputersystemen

Ein umfassendes Handbuch über die gesamte Mikrocomputer und Mikroprozessortechnik. Nach einer allgemeinen Einführung folgen Systembeschreibungen Z 80, sowie Hinweise für den Entwickler und Anwender. Ein Programmierbeispiel mit dem Z 80 Mikroprozessor Grundsystem runden dieses wertvolle Werk ab. Dieses Buch darf an keinem Arbeitsplatz im Labor und in der Entwicklung fehlen. Ca. 200 Seiten.
Lieferbar Sommer 1977 **Preis DM 29,80**

25

Hobby Computer Handbuch HCH, C. Lorenz

Die Computer und Mikroprozessortechnik dringt immer weiter in alle Lebensbereiche ein. Es ist soweit, daß auch der Amateur jetzt zu Hause seinen eigenen Computer zu einem erschwinglichen Preis selbst zusammenbauen kann. Welche Möglichkeiten es gibt, wie man ein solches System aufbaut, was man dazu benötigt u. vieles mehr, soll Ihnen dieses Buch zeigen. Komplette Systembeschreibungen, Grundlagen, Bedeutung für den Funkamateur, Versuchsschaltungen, Aufbau eines kompletten Systems, Hilfschaltungen u. v. a. mehr. Zentraleinheit. Speicherplatten 4k, 8k, 16k, Interfaceschaltungen, Displayinterface, Programmierung, Cassetteinterface, Teletype und andere Eingabeeinheiten. Ca. 150 Seiten.
Lieferbar Herbst 1977 **Preis DM 29,80**

26

Mikroprozessor Teil 2, H. Bernstein

Die Fortsetzung unseres ersten so erfolgreichen Buches über Mikroprozessoren. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, REPROM, FIFO, Schieberegister, MPR-Register, ARL-Register, SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM Schnittstellen. Befehlssatz 8080. Über 120 Seiten. Lieferbar Frühjahr 1977 **DM 19,80**

27

Mikroprozessor Software Handbuch MSH, C. Lorenz

Grundlagen und Einführung in die Mikroprogrammierung. Grundlagen und Einführung in die wichtigsten Programmiersprachen (BASIC, FORTRAN, ASSEMBLER-Sprachen) Zusammenstellung der wichtigsten Befehlslisten: 8080, Z80, M 6800, National, Fairchild, etc.
Ein Software Handbuch für jeden der mit Mikroprozessoren oder Mikrocomputern zu tun hat. Über 200 Seiten. Lieferbar Herbst 1977 **Preis DM 29,80**

28

Lexikon und Wörterbuch für Elektronik und Mikroprozessortechnik, LEM, C. Lorenz

Ein Hilfs- und Arbeitsbuch für jeden der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erklärung. Dieses Buch übersetzt Ihnen den englischen Fachausdruck und gibt Ihnen zusätzlich noch eine deutsche Erläuterung und Erklärung dieses Begriffes und was es damit auf sich hat. Ein Lexikon und Wörterbuch in einem einzigen Buch vereinigt. Das Buch, das Sie schon lange gesucht haben. Ca. 250 Seiten. Lieferbar Herbst 1977 **Preis DM 29,80**

29

Handbuch für theoretische und angewandte Elektronik, Teil 1, H. Bernstein

Eine umfassende Einführung in das gesamte Gebiet der Elektronik. Lehr- und Lernbuch zugleich. Grundlagenbeschreibung, Übungsaufgaben mit Lösungen, Rechenbeispiele, praktische Versuchsaufbauten mit Beschreibungen. Meß- und Prüfanleitungen. Das preiswerte Standardwerk wird Ihnen beim Selbstunterricht, als Hilfe für den Schulunterricht, als Nachschlagwerk und Handbuch ein wertvoller Begleiter durch die gesamte Elektronik sein. Umfang ca. 400 Seiten. Lieferbar Herbst 1977 **Preis DM 49,--**

30

Handbuch für theoretische und angewandte Elektronik, Teil 2, H. Bernstein

Fortsetzung der umfassenden Einführung in die gesamte Elektronik. Vom Elektron bis zur Rechentechnik.
Der zweite Teil führt von der Hochfrequenztechnik über Operationsverstärker und Boolsche Algebra bis zur Rechentechnik. Aufbau wie Teil 1 wieder mit genauen Einführungs- und Grundlagenbeschreibungen, sowie Rechenbeispielen und praktischen Versuchsaufbauten, Rechenbeispiele mit Lösungen etc. Ca. 400 Seiten. Lieferbar Frühjahr 1978
Preis DM 49,--

31

Basic Digital Electronics Course, C. Lorenz

Understanding Digital Electronics by using this „made simple" book and a brand new bredboarding system especially designed for this work book.
This book shows you how to use and design with integrated circuits. Number Systems, Logic fundamentals, TTL, CMOS Experiments: Logic Tester, electronic dice, counters, displays and many more fun projects for the experimenter.

200 pages Book alone **DM 19,80**
US $ 7,95

Bredboarding System WH–1g IC KIT **DM 79,--**
US $ 32,--

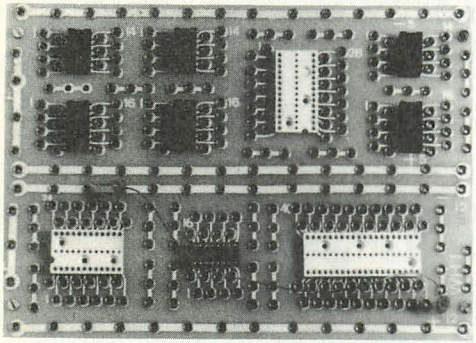
32

Circuits Digital Et Pratique, C. Lorenz

Dieses Buch ist eine französische Übersetzung unserer beiden Bücher Nr. 6 und Nr. 13 (IC-Schaltungen und Hobby Elektronik Handbuch HEH)
Ca. 100 Seiten

DM 19,80
FF 39,90

IC KIT Experimentier-platine



Mit der IC - KIT Experimentierplatine wird jetzt das Aufbauen von Schaltungen mit ICs, Transistoren, Widerständen und Kondensatoren zum Kinderspiel. Einfach, schnell und ohne Mühe lassen sich mit dieser hochwertigen Epoxy Versuchsplatine selbst die komplizierten Versuchsschaltungen aufbauen.
Die IC KIT Experimentierplatine Typ WH-1g eignet sich zum Aufbau von Uhren-, Rechner- und Mikroprozessorschaltungen besonders gut, da sie neben 14,16,24 und 28 poligen DIL - Sockeln auch einen 40 poligen Stecksockel enthält.
Der Einsatz von Adaptern ermöglicht auch das Stecken von diskreten Bauelementen. Weiterhin können Sie sich mit diesen Adaptern selbst kleine Funktionsgruppen aufbauen, die Sie immer wieder verwenden können, zum Beispiel Taktgeneratoren, Logiktester, Tastenentprellschaltungen etc. Der Schaltungsaufbau wird durch das Stecken aller Bauteile und Verbindungsleitungen wesentlich vereinfacht. Die einzelnen Elemente werden nicht beschädigt und bleiben frei von Lötzinn. Sie können immer wieder verwendet werden. Sie sparen Zeit und Geld !

41

Der Bausatz IC KIT WH-1g enthält: 1 Experimentierplatine in stabiler Epoxy - Ausführung mit den Abmessungen 210 x 150 mm fertig gebohrt und Leiterbahnen verzinnt. Die Bestückungsseite ist mit einem kratzfesten Lack überzogen und bedruckt. Sie können so die gesuchten Anschlüsse beim Experimentieren immer leicht und schnell finden. 340 Messingbuchsen, verschiedenfarbige Stecker mit Querloch (2,6 mm Ø), Metallfüße und sämtliche Sockel.

Bestell Nr. 41 — Preis DM 79.-

41/1

Adapter zum Stecken von diskreten Bauelementen, 5 Stück in einem Beutel verpackt. Die diskreten Bauteile werden hier einmal angelötet und können dann immer wieder verwendet werden. Einfachste Handhabung.

Bestell Nr. 41/1 — Preis DM 14,90

48

5V Netzgerät stabilisiert für die IC KIT Experimentierplatine Typ WH-1g. Bausatz komplett mit Netztransformator, Platine, allen Bauelementen, Netzkabel etc. Maximale Welligkeit 1% , TK = 0,5mV pro °C, Strombegrenzung. Maximaler Ausgangsstrom 650 mA. Ideal für alle TTL und CMOS Experimente.
Bestell Nr. 48 — Preis DM 39.-

Kurzlehrgang TTL CMOS

TTL - Kurzlehrgang
Der ideale Einführungs - kurs für das Selbststudium, den Unterricht , die Ausbildung und für den Amateur. Das Buch Nr,8 zusammen mit unserem IC - KIT Experimentierplatinenbausatz WH-1g bilden die ideale Grundlage für diese TTL -Einführung.
Kreuzen Sie auf der Bestellkarte einfach die Bestellnummern 8 und 41 an, Sie erhalten dann umgehend diesen wertvollen Kurzlehrgang zugeschickt.

8/41

Best.Nr.8/41 ÖS 889,20

CMOS -Kurzlehrgang
Der ideale Einführungs - kurs in die moderne C - MOS Technik. Grund - lagen, Entwurf, Versuchsschaltungen. Das Buch Nr.16 zusammen mit der IC-KIT Experimentier - platine WH-1g bilden die ideale Grundlage für diesen interessanten Kurs.
Kreuzen Sie auf der Bestellkarte einfach die Bestellnummern 16 und 41 an. Sier erhalten dann umgehend diesen C MOS - Kurs.

16/41

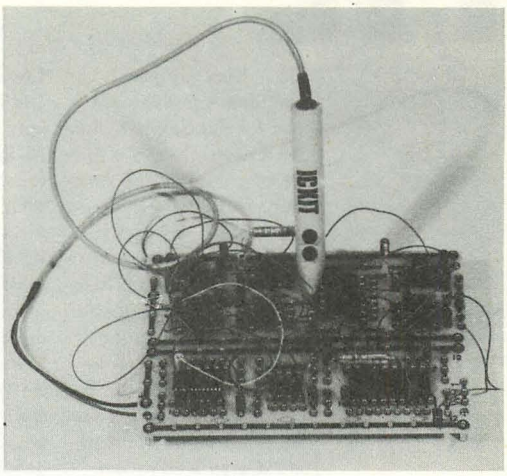
Best.Nr.16/41 ÖS 889,20

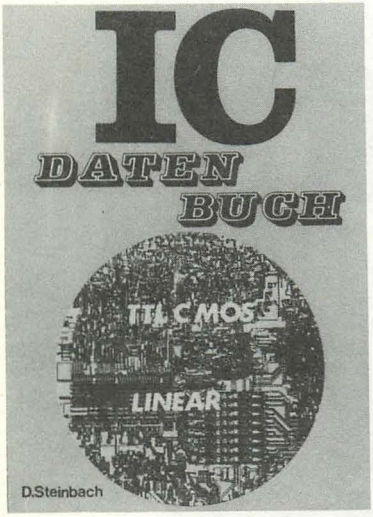
WIE GEHT'S WEITER?

Wie geht es weiter? Dies wird sich nun jeder nach dem Studium dieses kleinen Büchleins fragen. Wenn Sie Interesse an der Digitaltechnik gefunden haben, die ja zu den wichtigsten Grundlagen der Computertechnik gehört, können Sie sich anhand der nachfolgend aufgeführten Bücher leicht weiter in die Materie einarbeiten.

1. Digitaltechnik Grundkurs, Best.Nr.21
2. IC Experimentier Handb., Best.Nr.19
3. Beispiele integr.Schaltg., Best.Nr.12
4. IC Handbuch, TTL,C MOS, Best.Nr. 4

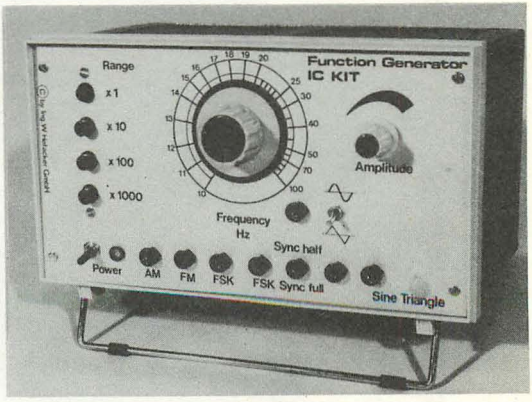
Rechts im Bild sehen Sie die Experimentierplatine, Best.Nr. 41 und den IC KIT Logiktester, den Sie als Bausatz oder Fertiggerät beim Fachhandel beziehen können.



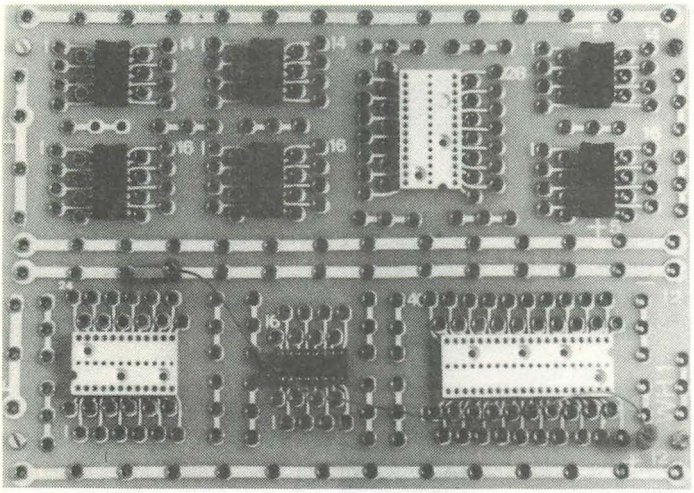


Im oben stehenden Bild sehen Sie das IC Datenbuch, ein wertvoller Helfer bei allen IC-Experimenten. Das Buch enthält Daten und Anschlüsse vieler wichtiger TTL, C MOS und Linearschaltungen. Best.Nr. 5, DM 9,80 erhältlich im Fachhandel oder direkt beim Verlag.

XR 2206 Funktionsgenerator
Dieser praktische und preiswerte Funktionsgenerator für Sinus-, Dreieck- und Rechteckwellen kann beim Fachhandel oder beim Verlag direkt als Bausatz oder Fertiggerät bezogen werden.
Frequenzbereich: 1 Hz bis 100 kHz
Niedriger Klirrfaktor: typisch 0,5 %
Hohe Frequenzstabilität: typisch $2 \times 10^{-5}/°C$
Alle Signalausgänge können frequenzmoduliert und amplitudenmoduliert werden. Auch eine Frequenzumtastung (FSK-Modulation) ist möglich.
Ausgangsimpedanz: Sinus/Dreieck=600Ω, Rechteck 10kΩ



IC KIT Experimentierplatte



Schaltungsaufbau durch Stecken der Bauteile, ohne zu Löten. IC -KIT Bausatz WH-1 g enthält alle erforderlichen Teile incl. Platte, Sockel, Buchsen, Stecker und Füße. Ideal für 40,28,24,16 und 14 polige DIL Geh.

IC - KIT Experimentierplatine TYP WH-1g
fertig aufgebaut. Maße: 210mm x 150 mm

Abs.

Name

Straße:

Ort:

Bestellkarte

Den Gesamtbetrag für die umseitig be - stellten Bücher bitte ich per Nachnahme zu berechnen/ habe ich heute auf Ihr Postscheckkonto München 15994-807 überwiesen. Nichtzutreffendes bitte streichen.

○ Bitte liefern Sie mir auch die Experimentierplatine IC -KIT WH-1g für 14,16,24,28 und 40 polige DIL IC-Gehäuse. Bausatz komplett : DM 79.-

○ Bitte liefern Sie von jedem neu erscheinenden Buch ein Muster per Nachnahme. (Rückgaberecht)

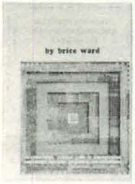
Ing.

W. Hofacker GmbH
Verlag
8 München 75
Postf. 437

Neue Bücher über Mikro. prozessoren

Modernste Elektronik Fachbücher in engl. Sprache über Mikroprozessoren, Mikrocomp. und Mikroprogrammierung.
Sofort lieferbar! Fragen Sie bei Ihrem Fach - oder Buchhändler, oder schreiben Sie an den Verlag.

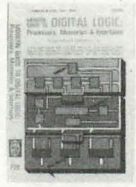
Alle Bücher sofort ab Lager lieferbar.



Microprocessor /Microprogramming Handbook.
Das praktische Handbuch für jeden der sich mit Mikroproz. beschäftigen will. Über 290 Seiten.
Best.Nr. 785 DM 35.-

Beginners Guide to Computer-Programming.
Die ideale Einführung in die Computerprogrammierung.
Gut geeignet für das Selbst - studium. Über 480 Seiten
Best.Nr. 574 DM 39.-

Computer Programming Handbook.
Eine genaue Einführung in die Programmierung und EDV mit ausgeführten Besispielen. Der ideale "Ein Buch Computer - kurs". Über 510 Seiten.
Best.Nr. 752 DM 45.-



Modern Guide to Digital Logic Processors, Memories, Interfaces

Speicherschaltungen, Interfaceschaltungen, ECL, SchottkyTTL u.v.a. mehr. Über 290 Seiten

Best.Nr. 709 DM 35.-

Master Handbook of 1001 practical Electronic Circuits.

Ein Schaltungsbuch mit über 1000 Schaltbeispielen mit kurzen Beschreibungen. Über 600 Seiten.
Best.Nr.800 DM 49.-

Best. Nr.	Stck.	BUCHTITEL BEZEICHNUNG	a DM	ges. DM
1		TBB – Handbuch Band 1	19.80	
2		TBB – Handbuch Band 2	19.80	
3		Elektronik im Auto	9.80	
4		IC – Handbuch, TTL, CMOS, Lin.	19.80	
5		IC – Datenbuch, TTL, CMOS, Lin.	9.80	
6		IC – Schaltungen, TTL, CMOS	9.80	
7		Elektronik Schaltungen	5.–	
8		IC – Bauanleitungshandbuch	19.80	
9		Feldeffekttransistoren	5.–	
10		Elektronik und Radio	5.–	
11		IC – NF – Verstärker	9.80	
12		Beispiele Integrierter Schaltungen	19.80	
13		HEH, Hobby Elektronik Handbuch	9.80	
14		IC – Vergleichsliste	9.80	
15		Optoelektronik Handbuch	19.80	
16		CMOS Teil 1, Einführung	19.80	
17		CMOS Teil 2, Schaltbeispiele	19.80	
18		CMOS Teil 3, Schaltbeispiele	19.80	
19		IC – Experimentier Handbuch	19.80	
20		Operationsverstärker	19.80	
21		Digitaltechnik Grundkurs	19.80	
22		Mikroprozessoren	19.80	
23		Elektronik Grundkurs	9.80	
24		Mikrocomputer Anwender Handbuch	29.80	
25		Hobby Computer Handbuch HCH	29.80	
26		Mikroprozessor Teil 2	19.80	
27		Mikroprozessor Software Handbuch	29.80	
28		Lexikon + Wörterbuch	29.80	
29		Handbuch für theoret. u. angew. Elektronik Teil 1	49. –	
30		Handbuch für theoret. u. angew. Elektronik Teil 2	49. –	
31 31		Basic Digital Electronics Course Circuits Digital Et Pratique	19.80 19.80	
41		IC–KIT Experimentierplatine WH–1g	79. –	

Notizen

Notizen

Verlagsprogramm

Übersicht lieferbarer Bücher

Bestell Nr.	ISBN	Verfasser	Titel	Preis DM
1	3-921682-01-0	Hofacker	Transistor Berechnungs- und Bauanleitungshand-buch, Band 1	19,80
2	3-921682-02-9	Hofacker	Transistor Berechnungs- und Bauanleitungshand-buch, Band 2	19,80
3	3-921682-03-7	Gebauer	Elektronik im Auto	9,80
4	3-921682-04-5	Lorenz	IC-Handbuch, TTL, CMOS, Linear	19,80
5	3-921682-05-3	Steinbach	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	3-921682-06-1	Steinbach	IC-Schaltungen, TTL, CMOS, Linear	9,80
7	3-921682-33-9	Hofacker	Elektronik Schaltungen	5,–
8	3-921682-08-8	Lorenz	IC-Bauanleitungs-Handbuch	19,80
9	3-921682-09-6	Lorenz	Feldeffekttransistoren	5,–
10	3-921682-34-7	Lorenz	Elektronik und Radio	5,–
11	3-921682-11-7	Lorenz	IC-NF Verstärker	9,80
12	3-921682-12-6	Bernstein	Beispiele Integrierter Schaltungen (BIS)	19,80
13	3-921682-13-4	Lorenz	HEH, Hobby Elektronik Handbuch	9,80
14	3-921682-14-2	Lorenz	IC-Vergleichsliste	9,80
15	3-921682-15-0	Lorenz	Optoelektronik Handbuch	19,80
16	3-921682-16-9	Bernstein	CMOS Teil 1 Einführung Entwurf, Schaltbeispiele	19,80
17	3-921682-17-7	Bernstein	CMOS Teil 2 Entwurf und Schaltbeispiele	19,80
18	3-921682-18-5	Bernstein	CMOS Teil 3 Entwurf und Schaltbeispiele	19,80
19	3-921682-19-3	Lorenz	IC-Experimentier Handbuch	19,80
20	3-921682-20-7	Lorenz	Operationsverstärker	19,80
21	3-921682-21-5	Lorenz	Digitaltechnik Grundkurs	19,80
22	3-921682-22-3	Bernstein	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	3-921682-23-1	Lorenz	Elektronik Grundkurs Kurzlehrgang Elektronik	9,80
24	3-921682-35-5	Hans Peter Blomeyer-Partenstein	Mikrocomputer-Anwender-Handbuch MAH	29,80
25	3-921682-25-8	C. Lorenz	Hobby Computer Handbuch HCH	29,80
26	3-921682-26-8	H. Bernstein	Mikroprozessor Teil 2	19,80
27	3-921682-27-4	C. Lorenz	Mikrocomputer Software Handbuch MSH	29,80
28	3-921682-28-2	C. Lorenz	Lexikon + Wörterbuch für Elektronik und Mikroprozessortechnik LEM	29,80
29	3-921682-29-0	H. Bernstein	Handbuch für theoretische und angewandte Elektronik Teil 1	49,80
30	3-921682-30-4	H. Bernstein	Handbuch für theoretische und angewandte Elektronik Teil 2	49,80
31	3-921682-31-2	C. Lorenz	Basic Digital Electronics Course (englisch)	19,80
31	3-921682-32-0	C. Lorenz	Circuits Digital Et Pratique (französisch)	19,80
41			Experimentierplatine für 14, 16, 24, 28, und 40 pol. DIL ICs.	79,–

Ing. W. Hofacker GmbH Verlag
8 München 75